COMPUTER LANGUAGE™

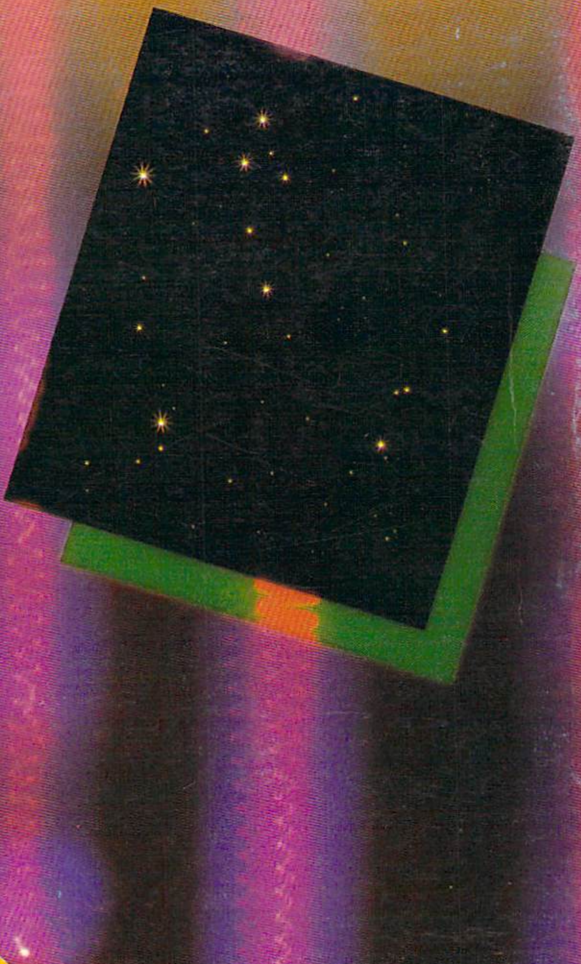
$2.95
VOLUME 1, NUMBER 2
OCTOBER 1984

THE EVOLUTION OF ZCPR

A PERSONAL VISIT WITH DONALD KNUTH

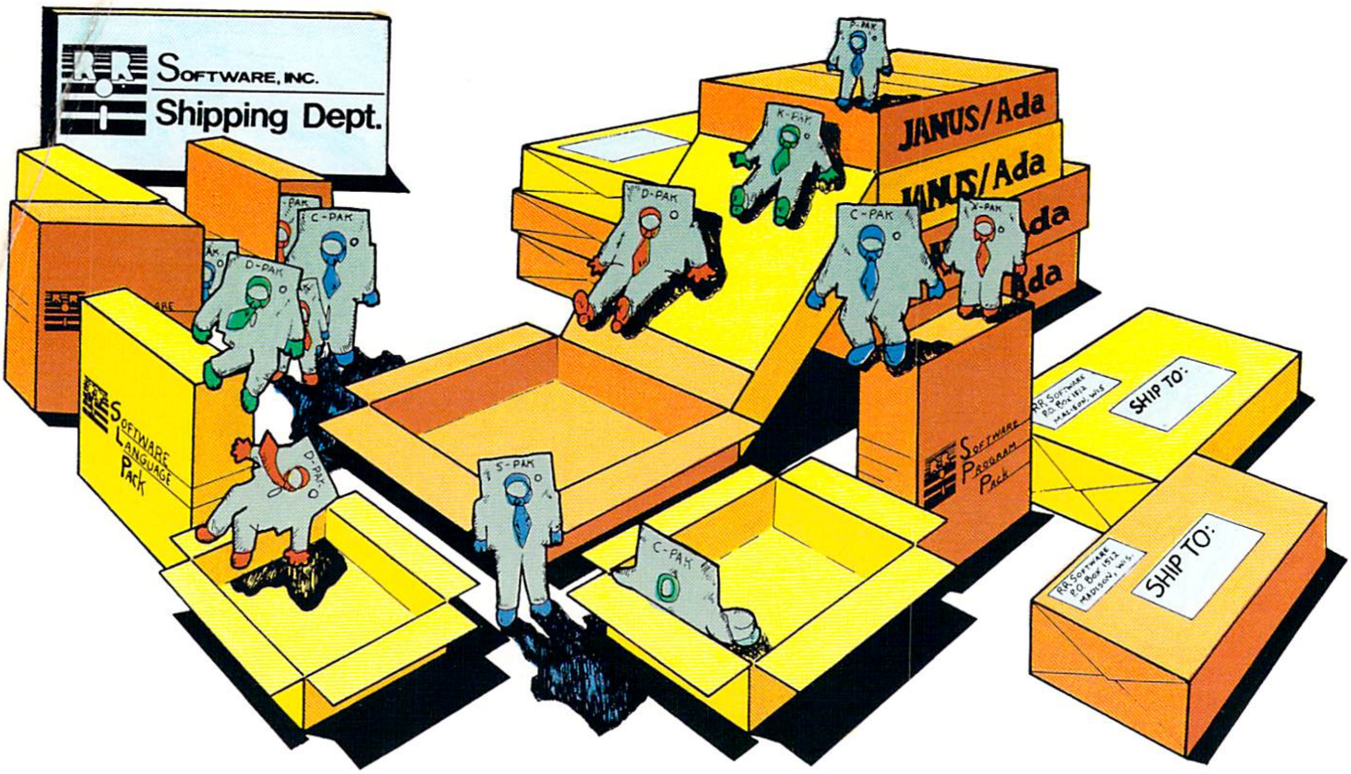
BATCH—A POWERFUL IBM "LANGUAGE"

COMPUTER LANGUAGE NOW ON COMPUSERVE!



PORTABILITY IN C
AN IMPLEMENTATION

WE'VE GOT YOUR PACKAGE!!



We offer you the most flexible, cost efficient means of introducing your programming staff to the Ada Language. **You** can choose the level of Support **you** need, when you need it! These Janus/Ada packages are customer-tested and available now...

- *(C-Pak) Introductory Janus/Ada Compilers*
- *(D-Pak) Intermediate Janus/Ada Systems*
- *(S-Pak) Advanced Janus/Ada Systems*
- *(P-Pak) Janus/Ada Language Translators*

- *Janus/Ada "Site" Licenses*
- *Janus/Ada Source Code Licenses*
- *Janus/Ada Cross Compilers*
- *Janus/Ada Maintenance Agreements*

Coming Soon: New Computer and Operating Systems Coverage

Selected Janus/Ada packages are available from the following:

National Distributors

Westico, Inc.
25 Van Zant St.
Norwalk, CT 06855
(203) 853-6880

Soft-Net
5177 Richard, Suite 635
Houston, TX 77056
(713) 933-1828

AOK Computers
816 Easley St., Suite 615
Silver Springs, MD 20910
(310) 588-8446

Trinity Solutions
5340 Thornwood Dr., Suite 102
San Jose, CA 95123
(408) 226-0170

Compuview Products, Inc.
1955 Pauline Blvd., Suite 200
Ann Arbor, MI 48103
(313) 996-1299

International Distributors

Micronix
11 Blackmore St.
Windsor 4030
QLD. Australia
(07) 57 9152

Progesco
155, rue du Fauburg
St. Denis
75010 Paris
(1) 205-39-47

Lifeboat of Japan
S- 13-14, Shiba
Minato-Ku
Tokyo 108 Japan
03-456-4101

CP/M, CP/M-86, CCP/M-86 are trademarks of Digital Research, Inc.
*ADA is a trademark of the U.S. Department of Defense
MS-DOS is a trademark of Microsoft

©Copyright 1983 RR Software

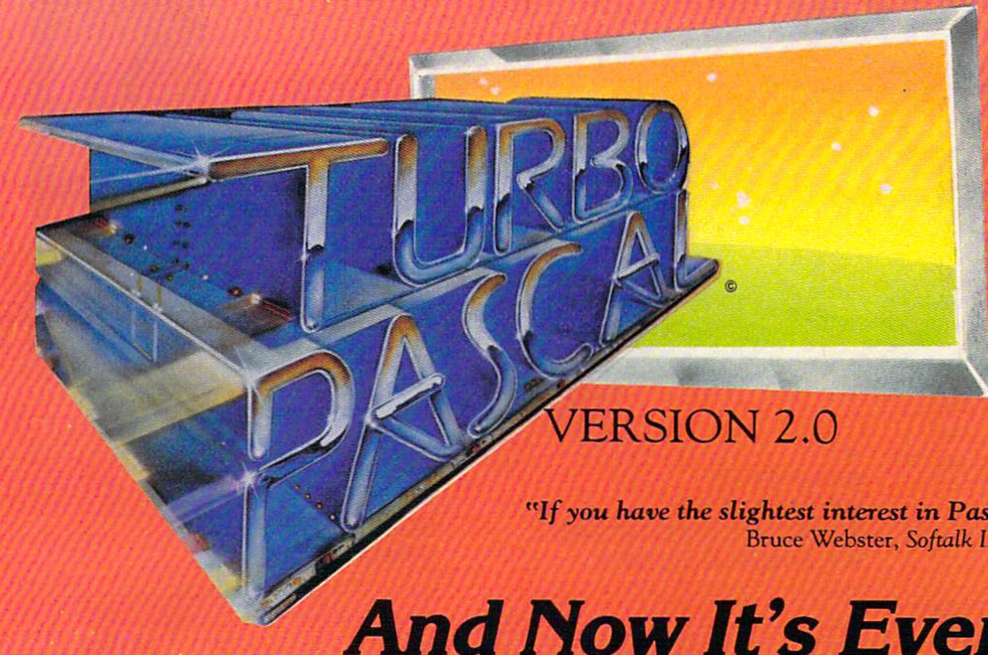
RR SOFTWARE, INC.

specialists in state of the art programming

P.O. Box 1512 Madison, Wisconsin 53701
(608) 244-6436 TELEX 4998168

CIRCLE 58 ON READER SERVICE CARD

This is THE PASCAL COMPILER You've Been Hearing About



VERSION 2.0

"It's almost certainly better than IBM's Pascal for the PC... Recommended."
Jerry Pournelle
Byte, May 1984

$49.95

"If you don't have CP/M [for your Apple], Turbo Pascal is reason enough to buy it."
Cary Hara
Softalk Apple, May 1984

"If you have the slightest interest in Pascal . . . buy it."
Bruce Webster, *Softalk IBM*, March, 1984

And Now It's Even Better Than You've Heard!

- **Windowing (IBM PC, XT, jr. or true compatibles)**
- **Color, Sound and Graphics Support (IBM PC, XT, jr. or true compatibles)**
- **Optional 8087 Support** *(available at an additional charge)*
- **Automatic Overlays**
- **A Full-Screen Editor that's even better than ever**
- **Full Heap Management—via dispose procedure**
- **Full Support of Operating System Facilities**
- **No license fees.** You can sell the programs you write with Turbo Pascal without extra cost.

Yes. We still include Microcalc . . . the sample spreadsheet written with Turbo Pascal. You can study the source code to learn how a spreadsheet is written . . . it's right on the disk.* And, if you're running Turbo Pascal with the 8087 option, you'll never have seen a spreadsheet calculate this fast before!

**Except Commodore 64 CP/M.*

Order Your Copy of TURBO PASCAL® VERSION 2.0 Today

For VISA and MasterCard orders call toll free: 1-800-255-8008
In California: 1-800-742-1133
(lines open 24 hrs, 7 days a week)
Dealer & Distributor Inquiries Welcome 408-438-8400

Choose One (please add $5.00 for shipping and handling for U.S. orders. Shipped UPS)

_____ Turbo Pascal 2.0 $49.95 + $5.00
_____ Turbo Pascal with 8087 support $89.95 + $5.00
_____ Update (1.0 to 2.0) Must be accompanied by the original master $29.95 + $5.00
_____ Update (1.0 to 8087) Must be accompanied by the original master $69.95 + $5.00

Check _____ Money Order _____
VISA _____ Master Card _____
Card #: _____
Exp. date: _____

BORLAND
INTERNATIONAL

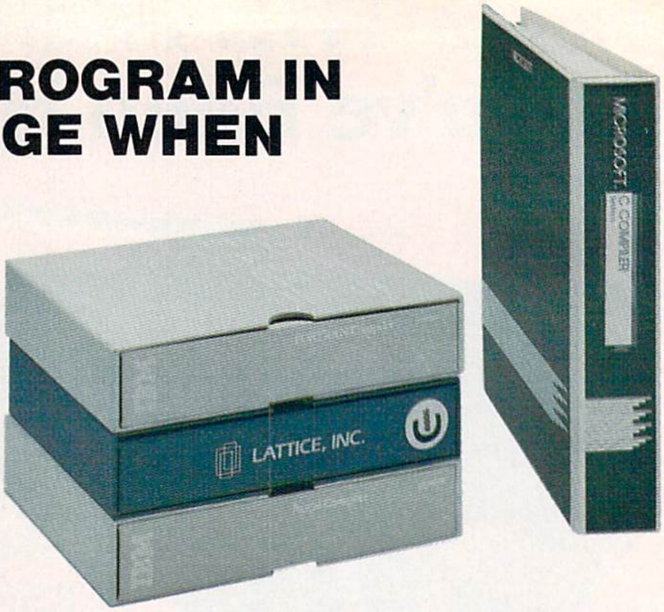
Borland International
4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX: 172373

CIRCLE 6 ON READER SERVICE CARD

My system is: 8 bit ___ 16 bit ___
Operating System: CP/M 80 ___
CP/M 86 ___ MS DOS ___ PC DOS ___
Computer: _____
Disk Format: _____
Please be sure model number & format are correct.
Name: _____
Address: _____
City/State/Zip: _____
Telephone: _____

California residents add 6% sales tax. Outside U.S.A. add $15.00 (If outside of U.S.A. payment must be by bank draft payable in the U.S. and in U.S. dollars.) Sorry, no C.O.D. or Purchase Orders. F20

WHY DEBUG YOUR PROGRAM IN ASSEMBLY LANGUAGE WHEN YOU WROTE IT IN ONE OF THESE...



ATRON Announces Source Level Software Debugging

Without source level debugging, the programmer must spend time mentally making translations between assembly language and the C, PASCAL, or FORTRAN source code in which the program was written. These tedious translations burn up valuable time which should be spent making critical product schedules. The low level hex and symbolic debuggers available today are superceded by ATRON'S solution — Source Probe.

HOW TO SINGLE STEP YOUR SOURCE CODE AND KEEP CRITICAL DATA IN VIEW

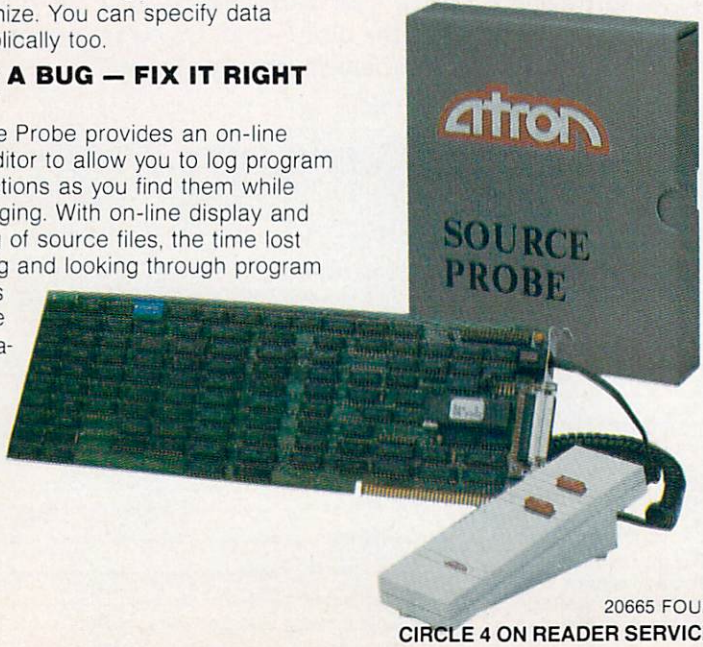
With Source Probe, you can step your program by source code statements. While stepping, a window which you define can display critical high level data structures in your program. The next several source code statements are also displayed to give you a preview of what the program will do

HOW TO DISPLAY DATA IN MEANINGFUL FORMATS

Why look at program data in hex when you defined it to be another data type in your program. Source Probe provides a formated print statement to make the display of your variables look like something you would recognize. You can specify data symbolically too.

FIND A BUG — FIX IT RIGHT NOW

Source Probe provides an on-line text editor to allow you to log program corrections as you find them while debugging. With on-line display and editing of source files, the time lost printing and looking through program listings can be eliminated.



A SNAP SHOT OF REAL TIME PROGRAM EXECUTION — BY SOURCE CODE !

When Source Probe is running on ATRON'S PC PROBE hardware, the real time execution of the program is saved. You can then view your source code as it executed in real time — including all the changes the program made to your data variables.

HOW TO FIND A BUG WHICH OVERWRITES MEMORY

When running on PC PROBE, the Source Probe can trap a bug which overwrites a memory location. Because complex pointers are normally used in high level language programming, this bug occurs frequently and is very difficult to find.

A BULLET PROOF DEBUGGER

What good is a debugger that can be wiped out by an undebugged program? With Source Probe running on PC PROBE, the software is write protected and cannot be changed.

ATRON PROVIDES THE DEBUGGING TOOLS WHICH FIT YOUR PROBLEM

PC PROBE —	A hardware aid to symbolic software debugging
SOFTWARE PROBE —	A symbolic debugger, runs without PC PROBE
SOURCE PROBE —	A source level debugger, versions run with or without PC PROBE
PERFORMANCE AND TIMING ANALYZER —	For finding where your program spends its time

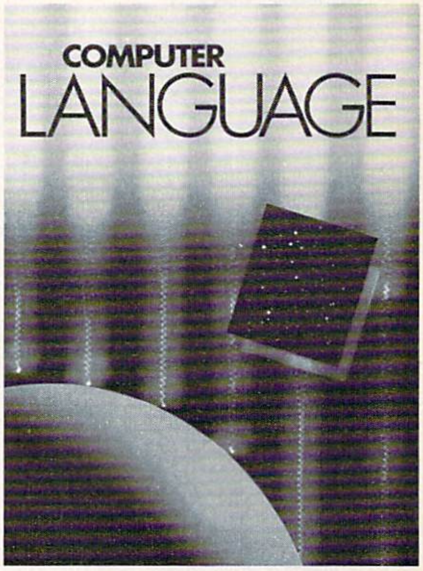
WE HAVE HUNDREDS OF HAPPY CUSTOMERS

ATRON produced the first symbolic debugger for the PC and the first hardware aided debugging tool — PC PROBE. We have hundreds of happy customers who have made their schedules because of ATRON debugging tools. Why waste more time — call us today!

atron
a debugging company
20665 FOURTH STREET • SARATOGA, CA 95070 • (408) 741-5900

CIRCLE 4 ON READER SERVICE CARD

COMPUTER
LANGUAGE
CL PUBLICATIONS



ARTICLES

An Implementation Demonstrating C Portability **21**
by David Harry, Ph.D.
C is often claimed to be the best language for writing code to be moved from one processor to another. Using one particular programming task as an example (writing a set of symbol table utilities that implement non-standard data structures), this author explores some of the realities involved in writing portably.

Batch—A powerful IBM "language" **31**
by Darryl E. Rubin
Many IBM PC owners have yet to discover that along with their PC-DOS 2.0 comes a powerful utility called Batch. Like many well established computer languages, this utility can perform procedure calling, recursion, argument passing, string manipulation, looping, case selection, and even file and console I/O.

The Evolution of ZCPR, Part I **37**
by Richard Conn
Over the past few years, CP/M programmers have been replacing the Console Command Processor (CCP) on their systems with a more powerful CCP replacement called ZCPR. In Part I of this two-part series, the founder of ZCPR—Richard Conn—reveals many of the powerful modifications he's made in the latest version, 3.0.

MNSNUS (or, Using Mnemonic Atoms in Symbolic Naming) **43**
by Ron Gutman
When writing long, complex code, it's often helpful to design a systematic method for naming things like variables and subroutines. The author of this article demonstrates a useful technique he's developed for structuring and organizing all user-defined symbolic names, regardless of the language used.

Low-level Assembly Interface on the IBM PC **47**
by Jeri Girard
Improve the speed and memory utilization of BASIC on the IBM PC by implementing some common assembly subroutines in a machine-level interface.

DEPARTMENTS

Editor's Notes ____ 5
Reader Feedback ____ 7
Back to the Drawing Board ____ 11
Designers Debate ____ 13
Creative programming vs. disciplined design
ComputerVisions ____ 17
A personal visit with the legendary Donald Knuth
Public Domain Software Review ____ 51
Exotic Language of the Month Club ____ 55
John Starkweather discusses the language he founded, PILOT.
The Code Swap Shop ____ 63
Software Reviews ____ 64
Telesoft's Ada, Volition's Modula-2, Catspaw's SNOBOL4+
Advertising Index ____ 80

We Do
Windows!

OPTIONAL
8087
SUPPORT

FORGET

EVERYTHING YOU THOUGHT YOU KNEW ABOUT PROGRAMMING IN BASIC.

introducing:

Better BASIC™

BetterBASIC offers:

- Support of large memory (to 640K).
- Extensibility (Make your own BASIC!!)
- Speed. Sieve of Erastosthenes Benchmark:
  - **BetterBASIC:** 31.9 seconds.
  - IBM PC BASIC: 191.1 seconds.
- Program Block Structures.
- User defined Procedures and Functions.
- Local and Global Variables.
- Shared Variables.
- Recursion.
- Argument type validation.
- Optional arguments.
- Arguments passed by-value or by-address.
- Separately compiled program Modules.
- Simple interface to Assembly Language Procedures.
- Support for OEM hardware through extensibility.
- Useful set of Data Types:
  - Byte, Integer
  - Real (variable precision BCD) Ideal for business math.
  - String (up to 32768 characters)
  - Record Variables & Structures.
  - N-dimensional Arrays of any type.
  - Arrays of Arrays.
  - Pointer (of any type)

"It combines the best points of interpreted Basic, Pascal, Forth and Assembler... It's the first piece of software I'd spend my own money on." Susan Glinert-Cole
Technical Editor
PC Tech Journal

We are so sure you will like **BetterBASIC**, we will give you a 30-day money-back guarantee. Order **BetterBASIC** now!
BetterBASIC: $199.00
8087 Module: $99.00
Not convinced? Then try the **BetterBASIC** Sample and you will find that **BetterBASIC** is truly a major breakthrough in computer programming.
Sample disk: $10.00

General Information:

- Interactive programming language based on an incremental compiler.
- Syntax checked immediately on entry, with concise error reporting.
- Built-in Screen Editor allows on-line editing.
- Full IBM Graphics/Communications Support.
- Built-in Linker for separately compiled program Modules.
- Built-in Cross Reference Lister
- Built-in WINDOWS support!!
- 8087 math support

Computer Requirements:

- IBM PC, IBM PC/XT or compatible.
- PC/DOS 1.1, 2.0, 2.1
- 192K to 640K memory
- Usable on plain MS-DOS machines with reduced functionality.
  (no Editor, Graphics or Windows)

OEM & Dealer Inquiries Invited.

BetterBASIC is a trademark of Summit Software Technology, Inc.
IBM PC, IBM PC/XT and PC/DOS are trademarks of International Business Machines Corp.
MS-DOS is a trademark of Microsoft Corp.

CALL YOUR DEALER OR SUMMIT SOFTWARE AT 617-235-0729

Summit Software Technology™ P.O. Box 99 Babson Park Wellesley, MA 02157

MasterCharge, Visa, P.O., Checks, Money Orders and C.O.D. accepted

CIRCLE 62 ON READER SERVICE CARD

UNIX TIMESHARE+

UNIX System III POWER and sophistication are yours. Let THE SOLUTION turn your micro into all you dreamed it could be, bringing the Ultimate programming environment as close as your modem. Just a local call from over 300 cities nationwide via Telenet.

THE SOLUTION™

- **EXPANSIVE SOFTWARE DEVELOPMENT FACILITIES** including Language and Operating System design.
- **LANGUAGES:** C, Fortran 77, RATFOR, COBOL, SNOBOL, BS, Assembler + Artificial Intelligence programming via LISP.
- **USENET Bulletin Board System**—800 + international UNIX sites feeding over 190 categories, typically bringing you more than 160 new articles per day.
- **Interuser and Intersystem mail** + 'chat' capability.
- **UNIFY:** Sophisticated data-base management system.
- **UNIX & System enhancements** from U.C. Berkeley and Korsmeyer Electronic Design Inc.
- **Online UNIX manuals** + Expert consultation available.
- **SOLUTION-MART:** Hardware/Software discount shopping database.
- **LOW COST and FAST** response time.
  (as low as $8.95 hr. connect time + $.05 cpu sec. non-prime)
- **$24.95 = 1 hr. FREE system time** + SOLUTION News subscription + BYTE BOOK (Introducing The UNIX System 556 pp.).

*UNIX is a trademark of Bell Labs.

Korsmeyer
ELECTRONIC DESIGN, INC.

5701 Prescott Avenue
Lincoln, NE 68506-5155
402/483-2238
10a-7p Central

Payment via VISA or Master Card

CIRCLE 34 ON READER SERVICE CARD

Editor's Notes

We need a bigger mailbox! When we started *COMPUTER LANGUAGE* six months ago, we had absolutely no idea that reader response would be so enthusiastic and supportive. I'd like to take this brief opportunity to thank all of you who wrote in with ideas, comments, and criticisms about our new magazine.

With this issue, *COMPUTER LANGUGE* goes monthly, incorporating all of the feedback we've received since the premier issue. Now that we know more accurately what readers want to see in *COMPUTER LANGUAGE*, we can begin to focus on those technical issues and trends that *you* have asked us to cover.

The big news this month is that you can now participate in an interactive, electronic *COMPUTER LANGUAGE* forum by dialing into the CompuServe Information Service and typing "GO CLM" (see advertisement on page 50).

The possibilities are endless! We're very excited about the concept of providing you with more information than any other printed magazine can offer. In addition to an electronic-mail-based reader forum, CompuServe gives us a way to start the largest public domain code swapping service ever, and all the material that couldn't fit in the magazine (like product reviews and program listings) will be there waiting for you.

We hope you'll use this electronic medium to keep us honest. We need and enjoy your feedback.

The *COMPUTER LANGUAGE* BBS (415-957-9370) has already become our own mini-CompuServe for those people who don't mind the long distance telephone call. So far, there are over 1,200 users on the system, answering their own electronic mail and downloading files. We'll soon be adding a hard disk to the system and will then have even more room for public domain code, software reviews, etc.

One special feature this month is *COMPUTER LANGUAGE*'s visit with Donald Knuth—the world famous Stanford University computer science professor and a man whose ideas have helped shaped our understanding of programming as we know it today.

Regarded as the world's top scholar in computer science, Donald Knuth is also an extraordinary mathematician, accomplished musician, composer, teacher, and inventor. Computer scientists today agree that Knuth's legendary *The Art of Computer Programming* series (published over 15 years ago) introduced order, clarity, and depth to a young, fragmented discipline.

In this month's *ComputerVisions* department, Knuth reveals some of his personal and professional thoughts about programming and life in general.

Next month, *COMPUTER LANGUAGE* visits with Gary Kildall, head of Digital Research Inc. and designer of the CP/M operating system.

Many people have written in suggesting that *COMPUTER LANGUAGE* sponsor a column for people who want to distribute programs and source code they've written. The new *Code Swap Shop* department is a place for people to show off their code to the readers of *COMPUTER LANGUAGE*.

If you've written a program that you would like to see distributed through our account on CompuServe or through our bulletin board computer, write us a note describing what your program is about and why other readers may be able to use it.

Once again, my hearty thanks for all the reader support we've gotten in the past three months. If you'll keep writing, we'll keep listening.

Craig LaGrow
Editor

COMPUTER LANGUAGE

EDITOR
Craig LaGrow

MANAGING EDITOR
Regina Starr Ridley

TECHNICAL EDITOR
John Halamka

EDITORIAL ASSISTANT
Hugh Byrne

CONTRIBUTING EDITORS
Burton Bhavisyat,
Tim Parker, Ken Takara

ADVERTISING SALES
Jan Dente

CIRCULATION COORDINATOR
Renato Sunico

ART DIRECTOR
Jeanne Schacht

COVER PHOTO
Dow Clement Photography

PRODUCTION/ART
Anne Doering

TECHNICAL CONSULTANT
Addison Sims

MARKETING CONSULTANT
Steve Rank

ACCOUNTING MANAGER
Lauren Kalkstein

CIRCULATION FULFILLMENT
Vincent Ridley, Mary Beth Faustine

PUBLISHER
Carl Landau

COMPUTER LANGUAGE *is published monthly by* COMPUTER LANGUAGE *Publishing Ltd., 131 Townsend St., San Francisco, CA 94107. (415) 957-9353.*

Advertising: For information on ad rates, deadlines, and placement, contact Carl Landau or Jan Dente at (415) 957-9353, or write to: COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

Editorial: Please address all letters and inquiries to: Craig LaGrow, Editor, COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

Subscriptions: Contact COMPUTER LANGUAGE, *Subscriptions Dept., 2443 Fillmore St., Suite 346, San Francisco, CA 94115. Single copy price: $2.95. Subscription prices: $24.00 per year (U.S.); $30.00 per year (Canada and Mexico). Subscription prices for outside the U.S., Canada, and Mexico: $36.00 (surface mail), $54.00 (air mail)—U.S. currency only. Please allow six weeks for new subscription service to begin.*

Postal information: Second-class postage rate is pending at San Francisco, CA and additional mailing offices.

Reprints: Copyright 1984 by COMPUTER LANGUAGE *Publishing Ltd. All rights reserved. Reproduction of material appearing in* COMPUTER LANGUAGE *is forbidden without written permission.*

Change of address: Please allow six weeks for change of address to take effect. POSTMASTER: Send change of address (Form 3579) to COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

COMPUTER LANGUAGE *is a registered trademark owned by the magazine's parent company, CL Publications. All material published in* COMPUTER LANGUAGE *is copyrighted © 1984 by CL Publications, Inc.*

THE FORTH SOURCE

MVP-FORTH

Stable - Transportable - Public Domain - Tools

You need two primary features in a software development package . . . a stable operating system and the ability to move programs easily and quickly to a variety of computers. MVP-FORTH gives you both these features and many extras. This public domain product includes an editor, FORTH assembler, tools, utilities and the vocabulary for the best selling book "Starting FORTH". The Programmer's Kit provides a complete FORTH for a number of computers. Other MVP-FORTH products will simplify the development of your applications.

MVP Books - A Series

- ☐ **Volume 1,** *All about FORTH* by Haydon. MVP-FORTH glossary with cross references to fig-FORTH, *Starting FORTH* and FORTH-79 Standard. 2nd Ed. $25
- ☐ **Volume 2,** *MVP-FORTH Assembly Source Code.* Includes CP/M®, IBM-PC®, and APPLE® listing for kernel $20
- ☐ **Volume 3,** *Floating Point Glossary* by Springer $10
- ☐ **Volume 4,** *Expert System* with source code by Park $25
- ☐ **Volume 5,** *File Management System* with interrupt security by Moreton $25

MVP-FORTH Software - A Transportable FORTH

- ☐ **MVP-FORTH Programmer's Kit** including disk, documentation, Volumes 1 & 2 of MVP-FORTH Series (*All About FORTH*, 2nd Ed. & Assembly Source Code), and *Starting FORTH*. Specify ☐ CP/M, ☐ CP/M 86, ☐ CP/M+, ☐ APPLE, ☐ IBM PC, ☐ MS-DOS, ☐ Osborne, ☐ Kaypro, ☐ H89/Z89, ☐ Z100, ☐ TI-PC, ☐ MicroDecisions, ☐ Northstar, NEW ☐ Compupro, ☐ Cromenco, ☐ DEC Rainbow, ☐ NEC 8201, ☐ TRS-80/100 $150
- ☐ **MVP-FORTH Cross Compiler** for CP/M Programmer's Kit. Generates headerless code for ROM or target CPU $300
- ☐ **MVP-FORTH Meta Compiler** for CP/M Programmer's kit. Use for applicatons on CP/M based computer. Includes public domain source $150
- ☐ **MVP-FORTH Fast Floating Point** Includes 9511 math chip on board with disks, documentation and enhanced virtual MVP-FORTH for Apple II, II+, and IIe. $450
- ☐ **MVP-FORTH Programming Aids** for CP/M, IBM or APPLE Programmer's Kit. Extremely useful tool for decompiling, callfinding, and translating. $200
- ☐ **MVP-FORTH PADS (Professional Application Development System)** for IBM PC, XT or PCjr or Apple II, II+ or IIe. An integrated system for customizing your FORTH programs and applications. The editor includes a bi-directional string search and is a word processor specially designed for fast development. PADS has almost triple the compile speed of most FORTH's and provides fast debugging techniques. Minimum size target systems are easy with or without heads. Virtual overlays can be compiled in object code. PADS is a true professional development system. Specify Computer. $500
- NEW ☐ **MVP-FORTH Floating Point & Matrix Math** for IBM or Apple $85
- NEW ☐ **MVP-FORTH Graphics Extension** for IBM or Apple $65
- NEW ☐ **MVP-FORTH MS-DOS** file interface for IBM PC PADS $80
- NEW ☐ **MVP-FORTH Expert System** for development of knowledge-based programs for Apple, IBM, or CP/M. $100

FORTH CROSS COMPILERS Allow extending, modifying and compiling for speed and memory savings, can also produce ROMable code. Specify CP/M, 8086, 68000, IBM, Z80, or Apple II, II+ $300

FORTH COMPUTER

- ☐ **Jupiter Ace** $150

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5 extra. Minimum order $15. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

FORTH DISKS

FORTH with editor, assembler, and manual.

- NEW ☐ **APPLE** by MM, 83 $100
- ☐ **APPLE** by Kuntze $90
- ☐ **ATARI®** valFORTH $60
- NEW ☐ **CP/M®** by MM, 83 $100
- ☐ **HP-85** by Lange $90
- ☐ **HP-75** by Cassady $150
- NEW ☐ **IBM-PC®** by LM, 83 $100
- ☐ **NOVA** by CCI 8" DS/DD$175
- ☐ **Z80** by LM, 83 NEW $100
- ☐ **8086/88** by LM, 83 NEW $100
- ☐ **68000** by LM, 83 NEW $250
- ☐ **VIC FORTH** by HES, VIC20 cartridge $50
- ☐ **C64** by HES Commodore 64 cartridge $60
- ☐ **Timex** by HW $25

Enhanced FORTH with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79, 83-FORTH-83.

- NEW ☐ **APPLE** by MM, F, G, & 83 $180
- ☐ **ATARI** by PNS, F,G, & X. $90
- NEW ☐ **CP/M** by MM, F & 83 $140
- ☐ **Apple, GraFORTH** by I $75
- ☐ **Multi-Tasking FORTH** by SL, CP/M, X & 79 $395
- ☐ **TRS-80/I or III** by MMS F, X, & 79 $130
- ☐ **Timex** by FD, tape G,X, & 79 $45
- ☐ **Victor 9000** by DE,G,X $150
- ☐ **C64** by ParSec. MVP, F, 79, NEW G & X $96
- ☐ **FDOS** for Atari FORTH's $40
- ☐ **Extensions** for LM Specify IBM, Z80, or 8086
  - ☐ Software Floating Point $100
  - ☐ 8087 Support (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (IBM-PC) $100
  - ☐ Data Base Management $200
- ☐ **fig-FORTH Programming Aids** for decompiling, callfinding, and translating. CP/M, IBM-PC, Z80, or Apple $200

FORTH MANUALS, GUIDES & DOCUMENTS

- ☐ **ALL ABOUT FORTH** by Haydon. See above. $25
- ☐ **FORTH Encyclopedia** by Derick & Baker $25
- NEW ☐ **The Complete FORTH** by Winfield $16
- ☐ **Understanding FORTH** by Reymann $3
- ☐ **FORTH Fundamentals,** Vol. I by McCabe $16
- NEW ☐ **FORTH Fundamentals,** Vol. II by McCabe $13
- NEW ☐ **FORTH Tools, Vol.1** by Anderson & Tracy $20
- ☐ **Beginning FORTH** by Chirlian $17
- ☐ **FORTH Encyclopedia Pocket Guide** $7
- ☐ **And So FORTH** by Huang. A college level text. $25
- ☐ **FORTH Programming** by Scanlon $17
- ☐ **FORTH on the ATARI** by E. Floegel $8
- ☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover) $18
- ☐ **Starting FORTH** (hard cover) $23
- ☐ **68000** fig-Forth with assembler $20
- ☐ **Jupiter** ACE Manual by Vickers $15
- ☐ **1980 FORML Proc.** $25
- ☐ **1981 FORML Proc** 2 Vol $40
- ☐ **1982 FORML Proc.** $25
- ☐ **1981 Rochester FORTH Proc.** $25
- ☐ **1982 Rochester FORTH Proc.** $25
- ☐ **1983 Rochester FORTH Proc.** $25
- ☐ **A Bibliography of FORTH References, 1st. Ed.** $15
- **The Journal of FORTH Application & Research**
  - ☐ **Vol. 1, No. 1** $20
  - NEW ☐ **Vol. 1, No. 2** $20
- ☐ **A FORTH Primer** $25
- ☐ **Threaded Interpretive Languages** $23
- ☐ **METAFORTH** by Cassady $30
- ☐ **Systems Guide to fig-FORTH** $25
- ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 User Man.** $20
- ☐ **FORTH-83 Standard** $15
- ☐ **FORTH-79 Standard** $15
- ☐ **FORTH-79 Standard Conversion** $10
- ☐ **Tiny Pascal fig-FORTH** $10
- ☐ **NOVA fig-FORTH** by CCI Source Listing $25
- ☐ **NOVA** by CCI User's Manual $25

☐ **Installation Manual for fig-FORTH,** $15

Source Listings of fig-FORTH, for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

☐ 1802 ☐ 6502 ☐ 6800 ☐ AlphaMicro
☐ 8080 ☐ 8086/88 ☐ 9900 ☐ APPLE II
☐ PACE ☐ 6809 ☐ NOVA ☐ PDP-11/LSI-11
☐ 68000 ☐ Eclipse ☐ VAX ☐ Z80 ☐ IBM NEW

MOUNTAIN VIEW PRESS, INC.

PO BOX 4656 MOUNTAIN VIEW, CA 94040 (415) 961-4103

CIRCLE 43 ON READER SERVICE CARD

FEEDBACK

A FORTRAN sort

Dear Editor:

Congratulations on an excellent first issue. As the vast majority of the material is worth reading, I won't use up space here trying to be too specific.

Richard Larson's article prompted me to follow up some of my thoughts on sorts. Certainly, his random number tests are fairer than previous articles, which are rigged to support the author's favorite algorithm. The random number generators supplied on many computer systems are questionable when used to support such comparisons (another subject for an article!).

I have tested sorts by odd tricks such as initializing the data to *x(i) = int(sin(float(i)))*, which gives a repeatable and possibly more representative test datum.

Larson's version of the Quicksort is the first readable one I have seen published and will save your readers some anguish. However, it is not designed for a FORTRAN-oriented machine, which most fast computers are. The main virtue is the ability to sort in place, which is done at the expense of code length. I have enclosed a two-step modification of the partitioning algorithm to show some alternative techniques (Listing 1).

My code performs redundant copies that are later overwritten by the correct data. Elimination of branching allows "optimizing" compilers and pipelining hardware to go to work. In the absence of these factors, the code will be significantly shorter than the original version but may take as much as 7% longer to run. Yes, we do need sorts in FORTRAN programs and they work!

Tim Prince
Marblehead, Mass.

```
Standard FORTRAN Quicksort Partition
(C uses scratch array TEMP for temporary copy of segment)

     Z = X(I)
     LEFT = 1
     RIGHT = J - I + 1
     DO 1 KP = I , J
          Y = X(KP)
          IF ( Y .GT. Z ) THEN
               TEMP(RIGHT) = Y
               RIGHT = RIGHT - 1
          ELSE
               TEMP(LEFT) = Y
               LEFT = LEFT + 1
          ENDIF
1    CONTINUE
     RIGHT = RIGHT + I - 1
     KT = 1
     DO 2 KP = I , J
          X(KP) = TEMP(KT)
          KT = KT + 1
2    CONTINUE
C Now X and Right are same as Larson's

Second Modification of Quicksort Partition replace DO 1

     for< kp=2 ; kp<=j ; kp = kp+1 ){
          y = x[kp]
          temp[left] = y
          temp[right] = y
          incleft =   y <= z
          left = incleft + left
          right = incleft - 1 + right
     }
     temp[right] = z
```

Listing 1.

Thanks and good luck

Dear Editor:

I do not remember ever writing a letter to the editor before, but I really wanted to say "thank you" for your first issue. I receive more than a dozen computer magazines. I write articles for two or three of them, but I have never before picked up a magazine and read it from cover to cover.

Every article in your premier issue compelled me to read it. I wish you the greatest of success and hope that success does not somehow spoil you. Keep it up!

Allen A. Watson
Hackensack, N.J.

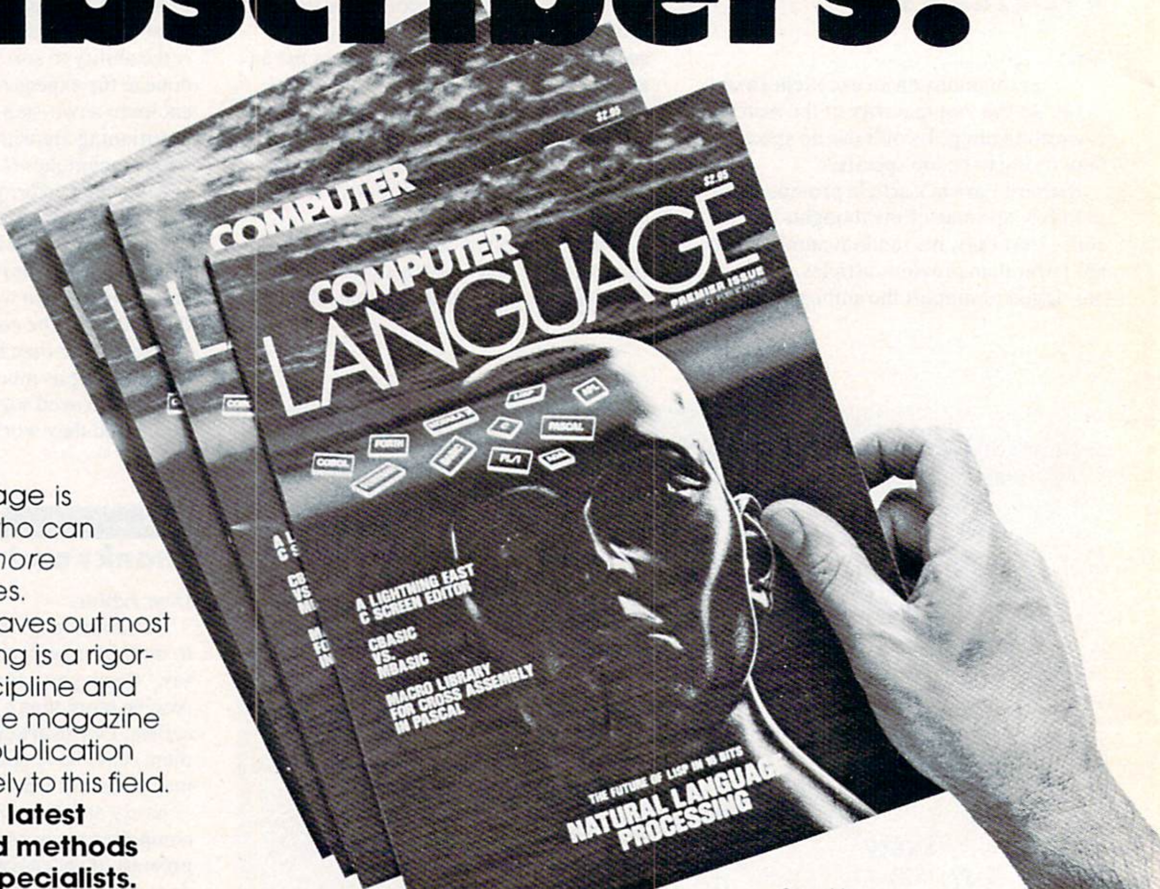
Forth rebuttal

Dear Editor:

Your column Designer's Debate: Forth vs. C compels me to write. While the proponents and moderator presented a great deal of information very well, some fundamental issues were missing, vague, and/or incorrect.

I've been a productivity proponent for my 15 years with computers. I've tried both languages as you suggested, as well as most other high-level languages, and numerous assemblers. A thrust of the modern languages (Pascal, Modula-2, Ada, UNIX, and C) is the detection of

We're looking for a few good subscribers.


$2.95
COMPUTER LANGUAGE
PREMIER ISSUE
COBOL
FORTH
MODULA-2
LISP
APL
PASCAL
C
BASIC
PL/I
ADA
A LIGHTNING FAST C SCREEN EDITOR
CBASIC VS. MBASIC
MACRO LIBRARY FOR CROSS ASSEMBLY IN PASCAL
THE FUTURE OF LISP IN 16 BITS
NATURAL LANGUAGE PROCESSING

Computer Language is written for people who can program in two or *more* computer languages.

Let's face it, that leaves out most people. Programming is a rigorous, intellectual discipline and Computer Language magazine is the first and only publication dedicated exclusively to this field.

Your source for the latest technical skills and methods used by software specialists.

We cover the major developments in the software design field, from theory to implementation. Computer Language focuses on the most important and useful language design information available in the fast-moving microcomputer industry.

Written for the person who takes computing seriously.

We're talking about you — the experienced software author, programmer, or engineer who routinely programs in two or more high-level languages. A person who understands the creative nature of programming and appreciates the beauty of efficient code in action.

COMPUTER LANGUAGE will constantly challenge your abilities.

The foremost industry experts will discuss: · Algorithmic Approaches to Problem Solving · Language Portability Features · Compiler Designs · Utilities · Artificial Intelligence · Editors · New Language Syntax · Telecommunications · Language Selection Criteria · Marketing Your Own Software · Critical Software & Hardware Reviews

Plus, columnists and reader forums that will put you in touch with the latest developments in the field.

Send to:
COMPUTER LANGUAGE
2443 Fillmore Street Suite #346
San Francisco, CA 94115

YES! Start my charter subscription to Computer Language. My 1 year charter subscription is just $19.95, a $15 savings under the single copy price. Guarantee: I can cancel my subscription at any time for a full refund.

☐ $19.95 Payment Enclosed ☐ Bill me.

Name

Address

City State Zip P

obviously incorrect programs. These languages validate a program by detecting errors just inserted by an on-line text editor. Thus errors are found immediately upon input, near the beginning of the project.

This thrust is misguided. There are more costly errors which these languages do not address.

More can be accomplished by avoiding time-wasters which crop up near project deadlines. When a project is late, testing is always sacrificed to more immediate matters, and quality suffers. In most languages, changing the specifications or package program can waste months, as can ill-conceived interfaces or misunderstood operating software. These problems are better addressed by Forth.

Why Forth? It is a clear channel of adequate expressive power. It is flexible. It is effective communication.

Forth fits the language to the problem. It is an extensible language, and various layers of the implementation can each be written in an appropriate notation. The notation can be optimized for readability and conformance to existing documentation. By demonstrating the top control code and other areas of interest with the user, project specifications can be agreed upon early; the project looks complete to the user before most of the work is done.

A pervasive benefit of Forth is speed. Development speed, final system speed, debugging speed, and response time speed. These all contribute to hardy interfaces. Interfaces can be discussed and tested from the beginning. During a meeting or review, the program can sometimes be changed on the spot to forge agreement.

Forth stimulates testing, retesting, and automated testing. Forth doesn't waste time, doesn't require short lunch breaks or several coffee breaks. In this respect, C is no better than FORTRAN, COBOL, or Pascal. They all take noticeable compile time. They all run only one program, lacking Forth's ability to combine or blend programs as needs dictate. Forth's flexibility stands on two legs: speed and the ability to blend programs.

A programmer who understands one portion of a Forth system can decipher the rest. One uniform language is employed for source code, copy statements, editor, preprocessor, job control language, linker control, automated test control, parameter libraries, symbolic debugger, device drivers, macro assembler, and operating system. Of course, when Forth is run on top of some host operating system, the host is a factor.

The Forth language is standard. The "different" versions of Forth are actually the same: FIG, PolyForth of Forth Inc., Forth-79, and Forth-83. They differ in their adaptation to different environments, such as varying sets of underlying operating system features or applications. The translation from one language version to another is practical, manageable, and straightforward, just like LISP translation. This is an advantage of extensibility. Future tinkering with the language may be an advantage to existing users because they won't be forced through the costly conversion process the COBOL shops face.

For example, I recently got a modem for my newest computer and began to transfuse information with an associate. We each keyed in and debugged a protocol of only nine lines of code. Your system would be the same, after making file assignments for *AKEY*, *PRINT2*, and *ME*. Being Forth to Forth, we quickly agreed to send blocks of 1,024 bytes. It was useful to take each line of code, just one definition or word, and test it exhaustively as it was entered. This verified the integration and understanding from the beginning. Within half an hour we were sending Forth screens of programs back and forth.

After lunch, my associate called back with a more elaborate protocol which allowed for unattended operation. It loaded into my machine without great effort. Now either machine can be controlled remotely, or control the other, or make remote file or program requests, or be a file server. Very standard, runs under Forth multitasking, and so on.

In Designer's Debate last month, it was noted that C allows the programmer to specify register optimizations. C programmers *have* observed improved program speed with this feature. This type of optimization . . . is regaining favor.

Forth supports a whole spectrum of optimizations. It is a myth that Forth programs are inherently slow. When a program is nearly complete, and speed is the issue, then maximum machine speed can be crafted with a choice of documented techniques, such as assembling those few lines of code within inner loops. Forth also supports local variables, either by name or absolute stack location.

An important Forth advantage is its development speed, and hence its ability to deal with unknowns. UNIX and C apply to fixed hardware configurations, as on VAX or AT&T hardware. Forth applies well to custom hardware or the integration of boards from different manufacturers. Thus it can be found in embedded controllers, instruments, automated testing equipment, and spacecraft.

Forth screens seem incomprehensible to some, but I believe screens are superior. Screens encourage the creation of small comprehensible programs. Each screen can be accessed, compiled, or sur-

YOUR CODE MAY BE WASTING ITS TIME!
THE PROFILER™ CAN HELP . . .

- Statistical Execution Profiler
- Works with any language
- Completely configurable
- Up to 16 partitions in RAM/ROM
- Time critical code optimization
- Abnormal code behavior tracking
- Graphic presentation of results
- Easy to use menu interface

THE PROFILER is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With the **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER**, you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

THE PROFILER is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

NO ADDITIONAL HARDWARE IS REQUIRED!

Requires an IBM PC or compatible system with a minimum 64k and one drive.

THE PROFILER is available for $175.00 from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
THE PROFILER is a trademark of DWB Associates.

dwb
Associates
P.O. Box 5777
Beaverton, Oregon 97006
(503) 629-9645

CIRCLE 20 ON READER SERVICE CARD

APC
MEGABASIC

8086/8
CP/M-86
MP/M-86
MS-DOS
TURBODOS

MEGABASIC™ reduces program development time and memory requirements dramatically, executes up to 6 times faster than MBASIC interpreter, is highly portable among virtually all microcomputers, and is supported by outstanding documentation.

BENEFITS:
- Large Memory—Up to 1 Mb programs and data.
- Fast execution—as fast as many compilers.
- Easy program development—advanced TRACE and EDIT functions.
- Rounding errors eliminated—BCD arithmetic.
- Simple to use—No complicated field statements.
- Source code protection—"scramble" utility.

THE COMPLETE PACKAGE:
—Developmental version of MEGABASIC in precisions up to 18 digits.
—Run-time semi-compiler version.
—Compaction utility reduces program size.
—Cross-reference generator that lists all variables, arrays, subroutines, functions, etc.
—Function library with fast sorts, yes/no prompt routines, matrix manipulation and many more routines ready to plug into your progams.
—Configuration program.
—350 page manual with more than 2,000 index entries.
Complete package: $400
Dealer inquiries invited.
VISA or MasterCard accepted.

AMERICAN
PLANNING
CORPORATION
4600 Duke St.
Suite 425
Alexandria, VA 22304
1-800-368-2248
(In Virginia, 1-703-751-2574)

CIRCLE 3 ON READER SERVICE CARD

veyed by the eye in a second. The editor begins and responds in less than 1 sec. Data storage is provided for, as is a faster editor, a file system, a fast idea-edit-compile-test-think cycle, hence quick product development.

I don't know where I'd be without Forth.

Gary Nemeth
Cleveland, Ohio

Keeping us honest

Dear Editor:

I really like your new magazine, but we've been in the business an aggregate of 30 years and have seen magazines come, get good, and then turn into expensive books of advertising. So far, you've thrown out the power incentive to us, the readers, to keep *you* honest—we like that. This BBS is just one example of that kind of incentive.

M.K. Sargent
New Jersey

A language junky

Dear Editor:

At last: a magazine for all us language junkies! I thought your first issue was great.

Keep it up and bring on the weird languages (especially liked your SNOBOL piece).

Doug Clapp
InfoWorld

COBOL attack

Dear Editor:

I would like to say a few words about Robert Wagner's article, "COBOL: Pride and Prejudice."

Why do companies prefer COBOL, Wagner asks? Could the answer simply be that COBOL is the most *maintainable* language, based upon traditional prejudice, and that it is the only language which management knows?

Let us not forget that most of the DP managers and others went through the system when COBOL programs were on punch cards. Wagner attempts to support the maintainability theory by presenting three programs written in COBOL, C, and Forth.

COBOL programs may be easy to read but that does not mean they are easy to understand. I had to desk check the COBOL program in the article twice before I was able to understand what the program was doing. Let us consider something simple, the statement *MUTIPLY A BY B*. One would expect *A* to be multiplied by *B* and the result stored in *A*. This is the exact opposite of what happens.

Wagner points with pride to the statement:

```
MOVE ZEROS TO PRIME-COUNT,
    FLAGS
```

and proclaims "aren't we clever and fast?". I am not impressed. Observe the same programming logic in APL:

```
PRIMECOUNT < – +/FLAGS < –
    8191$RO0
```

Now, in BASIC:

```
DIM FLAGS(8191)
```

where *PRIMECOUNT* has a default value of 0.

In Pascal,

```
FILLCHAR(FLAGS,8191,CHR(0));
PRIME_COUNT := 0;
```

I know of only one language that took anything from COBOL—PL/I. If COBOL is as great as Wagner wants us to believe, why haven't more new languages taken ideas from COBOL?

I have to disagree with Mr. Wagner's opinion that "big people" will become an important factor on the PC scene in two or three years. If what I have been reading about the corporate resistance to the PC is correct, it will take a major corporate revolution for this to happen.

Finally, Mr. Wagner claims that COBOL is technical enough to do anything and informal enough to be easy to understand. The program segment presented with the article does not support these claims. My own view is that COBOL is not suitable for most programming tasks.

Keith A. Van Wagner II
Dundee, N.Y.

Problem solving

Dear Editor:

Yes, I think that there is a need for a magazine for people who need languages to solve a broad range of problems. Perhaps if more people who were engaged in problem solving were to become involved in language development, we would be beyond the fifth and up to some succeeding generation by now (if you believe that generation has any meaning to begin with).

Lynn Maxson

BACK TO THE DRAWING BOARD

High tech, high touch

By Burton Bhavisyat

I just got through reading John Naisbitt's *MegaTrends*, and I'm inspired and impressed.

I'm not as much impressed with the conclusions he draws or the social, political, or economic trends he reports as with the way he got his facts.

Content analysis. His outfit monitored 6,000 local newspapers every month and analyzed the action to isolate current trends. His book is a report on the trends of the past few years with some predictions about the future, based on real, down-to-earth facts.

I don't really need to tell you that he found the U.S. is growing away from being an industrial society and toward becoming an information society. Not exactly hot news, right? But he also has a few things to say that are not so obvious.

He makes the point that whenever a new technology is introduced into a society, there must be a counterbalancing human response—that is, "high touch"—or the technology is rejected by the members of that society. The more high tech, the more high touch.

With the preceding in mind, I'd like to introduce Bhavisyat's way to bring some high touch to the high-tech world of *COMPUTER LANGUAGE*.

Remember those times when you threw up your hands in frustration trying to figure out just what the *%$ is going on with a particular programming problem that you've been wrestling with? Wouldn't you have loved to be able to call upon a friend who just so happened to be an expert in the field?

Now you can. Let me explain . . .

Back to the Drawing Board is meant to be a reader-feedback column. And indeed it is. But more than that, we've got the facility and opportunity to do a lot more than just send letters to one another.

COMPUTER LANGUAGE has a Bulletin Board Service set up and a special account on CompuServe that serves the same function but in a more national context (i.e., local phone bills!).

These systems let you communicate directly with the editors of *COMPUTER LANGUAGE* and the authors of its articles and departments. That's high touch! If you haven't tried it yet, please do—you'll have a pleasant surprise.

In the first two weeks after the premier issue of *COMPUTER LANGUAGE* was released, over 900 people called the BBS. And, as of this second issue, CompuServe is ready for those people hesitant to make the long-distance computer call to San Francisco, Calif. We've got even bigger ideas in store for the future of our own electronic village, so watch and see.

I'm sure you've seen in other mags those columns whose writers answer questions submitted by readers. Haven't you noticed that, in some cases, you could have done a better job answering those questions than the author writing the column? At *COMPUTER LANGUAGE*, we'll give you the chance to do just that.

The old fashioned process is when readers send me a letter with their questions, and I select some people from our expert audience to take a shot at answering. This works (and we want to keep on doing it), but it's slow. Here's a quicker way . . .

If you think you're knowledgeable in some area, and you might enjoy answering questions posed by people who are eager to know what you know, either leave a note on the BBS message system addressed to me, or send me a letter with your name and phone number (or a mailing address, if you'd prefer to get feedback that way). Include a list of what areas you're up on, and I'll get together a big list of such persons, sorted by state and expertise.

This listing will be available on the BBS (and occasionally in *COMPUTER LANGUAGE*). If anyone has a nagging problem in your field, they can get in touch with you to ask you. This could easily become a fun way of getting new ideas from curious people as well as a way to help people over their own technical hurdles.

Alternatively, if *you* have a problem, you'll know where to go to for help.

That's not the end of it, either. Since *COMPUTER LANGUAGE* is now available on CompuServe, you will soon be able to get immediate on-line help with your problems (especially if you include your CompuServe handle in your address data).

To get involved, dial up the *COMPUTER LANGUAGE* BBS at (415) 957-9370 or call your local CompuServe node and use the message system to send your mail to Burton Bhavisyat. Or, if you like, you can send the same information in a letter to *COMPUTER LANGUAGE*, c/o Bhavisyat, 131 Townsend St., San Francisco, Calif. 94107.

I've already got my expert list going, and you'll see it when you call in. You'll also be watching it grow over the coming months and years.

High tech, high touch. Let's go for it!

Multi-Basic

"The BASIC compiler that compiles both MBASIC and CBASIC"

Now you don't have to give up the features you like about MBASIC to obtain the powerful capabilities of CBASIC. Multi-Basic gives you both.

Multi-Basic works with your existing programs so your current software investment is protected. But just as important, Multi-Basic opens the door to a whole new way of programming. With Multi-Basic you can write very readable, modular and structured programs. Multi-Basic makes program maintenance as easy as it is with Pascal.

In addition to understanding the two most popular dialects of BASIC, Multi-Basic allows you to extend the language even further. You can add your own statements and functions as needed.

Multi-Basic is also compatible with our Pascal and C compilers. This allows your BASIC programs to use routines written in Pascal or C.

In today's fast changing computer business, you need a language as versatile as Multi-Basic. Invest a little time today and save a lot of time tomorrow. You owe it to yourself to see what a difference Multi-Basic can make.

Multi-Basic is available for the TRS80 models I, II, III, 4 and 12; Tandy 2000, IBM PC, and CP/M. It is compatible with TRSDOS, LDOS, NEWDOS, DOSPLUS, MSDOS, PCDOS, CP/M and CP/M plus.

Alcor Multi-Basic	$139
Other Products:	
Advanced Development Package	$ 69
Blaise I Text Editor (Mod 1 or 3)	$ 49
Blaise II Text Editor (all others)	$ 79
Multiprocessor Assembler	$ 69
Alcor C	$139
Alcor Pascal (for CP/M, MSDOS, PCDOS)	$139
Complete Development System includes compiler, text editor and advanced development package	$250
Shipping U.S.A.	$6.00
Shipping Overseas	$28.00

ALCOR Systems

13534 Preston Road, Suite 365
Dallas, Texas 75240
(214) 494-1316

Multi-Basic is a trademark of Alcor Systems
TRS80 is a registered trademark of Tandy Corporation
CP/M, CBASIC are trademarks of Digital Research
MSDOS, MBASIC are trademarks of Microsoft

CIRCLE 1 ON READER SERVICE CARD

CONFUSED?

Communications Software Can Be a Real Headache. For FAST RELIEF, use COMMX! It's Simple to Operate and Provides the Best Features Available for Both Personal and Business Communications:

- Easy to Use Menu Selections and Prompts
- Auto-Dial-Logon and Unattended Controls
- Dial Directory Handles up to 700 entries
- Install Utility for Intelligent Modems
- Programmable Terminal Emulation!
- Linkup with Information Services like WU Telex, TWX, USPS ECOM, CompuServ, NewsNet (free subscription included)
- Micro to Micro and Micro to Mainframe multiple File Transfer Protocols:
  - Text Upload/Download with Options
  - Text and Binary Upload/Download with proprietary Error-free COMMX protocol mainframe Versions available for VAX, CompuServe, DEC 10, IBM 370, HP3000, PRIME
  - MODEM7 Batch and Single file Send/Recv
- Direct Link High Speed Data Transfers
- Electronic Mail Management Software upgrade Available for Organizations
- InfoWorld Report Card A + + + + Dec 1981

COMMX is priced from $195 (micro CP/M or MS-DOS) to $900 (mainframe). OEM and multiple licenses available.

HAWKEYE GRAFIX Inc ™
818-348-7909 / 213-634-0733
23914 Mobile, Canoga Park, CA 91307

CIRCLE 30 ON READER SERVICE CARD

DESIGNERS DEBATE

Creative programming vs. disciplined design

By Ken Takara

As programmers, many of us have certain reservations about the subject of programming itself. We are warned about the dangers of "hacking," yet many of us have suffered through long and unproductive design sessions where several vociferous individuals have dictated the course of a software project despite their obvious ignorance of the abilities and limitations of the computer.

In this month's debate, Peter Nau, software engineer, defends the design process against some of the more common criticisms. It is hoped that some misconceptions about program design may be clarified. On the other hand, many misconceptions about hacking still linger. In an upcoming column we'll look at the fine art of hacking and try to obviate the numerous misunderstandings that have evolved there also.

Q What is hacking?

A Well, to me, hacking is when you throw something together then try to debug it until it works. It's useful when you're exploring—in fact, it's an excellent way to try something out just to see how it works. You can learn a lot that way. It can also be useful sometimes when you're trying, for example, to improve a subroutine or a small module. At some point, everyone ends up doing some hacking to make that piece of code work a bit better.

The problem with hacking is that it's a local activity. You generally take a very restricted view of what you're doing. It's not conducive to looking at the global scope of the project. When you design, you approach the project as a whole, which you plan out before actually trying to build anything. Imagine a carpenter nailing beams together to build a house without having some sort of plan to start from.

Software development is analogous to hardware design. I once knew a hardware hacker who "designed" by sketching out what he thought a circuit should look like. Then he wired the components together and interactively tested and modified it until it worked.

Sure, eventually it ran, but it was impossible to figure out what it was doing. You never knew what that thing hanging on the side did, but if you removed it, the circuit would cease to operate. Of course, he learned a lot by doing it this way, but it was often difficult to fix it when it failed.

Q What does software design involve?

A Software design is a method by which you take a problem, analyze it, then build a software solution. There are several steps you typically go through. The first thing is to find out what the problem is—this is what systems analysts do. Then, assuming you're going to write a program, you specify its requirements. You then design and structure the program, dry testing it. The last thing you do is build it.

When someone asks for a program, you usually start by interviewing them. You want to understand the nature of the problem—first by starting at a general level, then getting into more detail. Often while I'm trying to understand the problem, I draw diagrams. It's easier for me to think pictorially, and I can show it to the requester and ask, "Is this how it works?"

Q Suppose you are writing the program for yourself. Do you really need to go through all this interviewing and diagramming? Wouldn't a programmer have a pretty good idea what he or she wants as well as how to get there already?

A That sort of thing can really bring out the hacker in me! Yes, I know what I want and can think of all sorts of things to add to it and various ways to make parts of it particularly clean and efficient. This really gets into the nature of my own creative process. What I do is "interview" myself and keep notes on all the ideas I come up with along the way. It's important to know what I really want and what is just a nice frill. It becomes a matter of creative energy vs. discipline. Maybe you can think of a hacker as a person full of creative energy but lacking discipline.

By the way, there is a fine art to trimming off the non-essentials that you initially imagine are so important. I find it interesting that the non-essentials I scrub seldom come back. Of course, I try to design so that I can put them in later if I decide I want them.

Q What kinds of diagrams do you like to use?

A I've been using dataflow diagrams (Figure 1) during analysis. They're useful whenever you are following the flow and transformation of information—as in business software or in industrial automation, where you follow actual physical entities. Architecture diagrams (Figure 2) are good for designing asynchronous tasks. The semaphores can be treated as intertask messages; tasks talk to each other either to send information or to pass control.

I sometimes use state diagrams (Figure 3) when designing the logic of modules that control sequential machines. And structure charts (Figure 4) are useful to show the calling relationships between modules in a single-task environment. Then there are such things as the Warnier/Orr diagrams, HIPO (Hierarchical Input-Process-Output) charts, and so on.

Q I remember a fellow who would get into complex discussions on the importance of rounded corners on the process boxes of data flow diagrams. I thought that this was quite overdone. Just how important are these diagrams?

A The whole point of the diagram is to provide a picture of your understanding of the problem, and of how the system works or is supposed to work. It is just a step in the transformation from problem to solution, not an end in itself. It is more important for the diagram to be consistent and clear.

Once the analysis is completed, you should be able to come up with a set of requirements that specify what the program should do. You design the program so that it will meet these requirements. You may also want to have a list of desirables that you might consider for inclusion once the requirements are met. Specifying the requirements can be tricky because you don't want to over-specify them.

Q I have seen cases where the specifications got into the implementation detail. This brings up another set of questions. In *The Mythical Man Month* (published by Addison-Wesley, copyright 1982), Frederick Brooks suggests having an "architect" who is responsible for design. Is it such a good idea having a designer who doesn't code or who is out of touch with the nature of the computer?

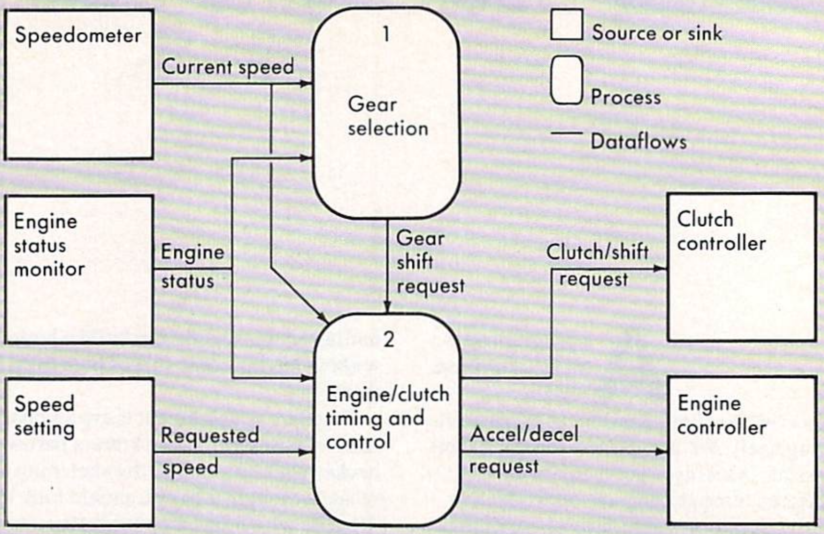
A Well, the designer definitely should be familiar with the medium. Frank Lloyd Wright would visit the site on which he was designing a building. He wanted to keep in touch with the medium with which he dealt. I remember a designer who insisted on designing for a "virtual" machine. His designs were quite awkward for the programmer.

Brooks also suggests keeping the duties of programmers and designers separate. One engineer I know found the idea practically dictatorial—an autocracy of architects with coders slaving away doing just grunt work.

I certainly wouldn't want to be a programmer who just cranked out code. Maybe there are people who wouldn't mind it. Perhaps we should have software technicians, just as there are hardware technicians who just put circuit boards together. Anyway, Brooks was suggesting a solution to the problem of organizing complex design projects. I think he is essentially correct, though his approach may be a bit simplistic.

I think what Brooks wants is design coherence or conceptual integrity. Having

Dataflow diagram


Speedometer
Current speed
1
Gear selection
Source or sink
Process
Dataflows
Engine status monitor
Engine status
Gear shift request
Clutch/shift request
Clutch controller
2
Engine/clutch timing and control
Speed setting
Requested speed
Accel/decel request
Engine controller

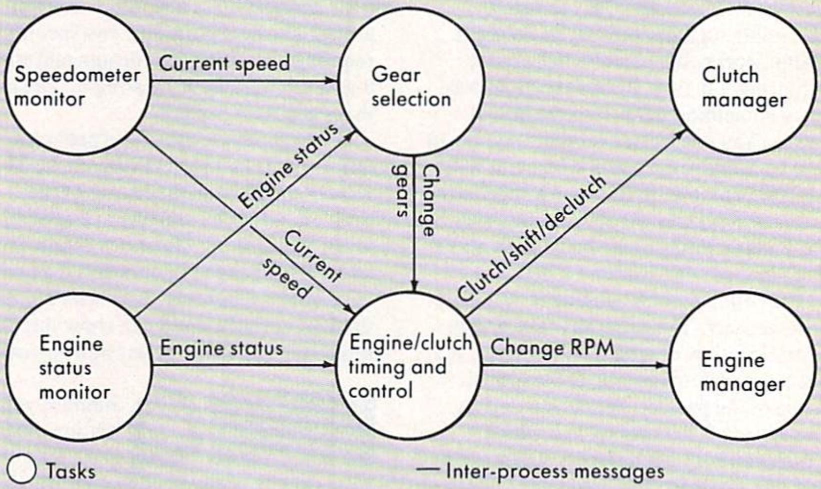
This dataflow diagram is for an automobile cruise control system. The square boxes along the perimeter of the diagram represent entities external to the system. The system of interest to us consists of the two process boxes (with rounded corners) and the set of connecting dataflows. The sources and sinks, external to our system, remain as "black boxes" to us.

Process 1 (gear selection) gets the current speed and engine status and tries to select the appropriate gear. If a gear change is desired, it sends this information to Process 2. For example, if the engine is laboring, it may request a downshift. Process 2 (engine/clutch timing and control) gets current speed and engine status as well as the requested speed, which it attempts to maintain by correcting the RPM. It also tries to prevent engine laboring by reducing RPM if necessary. If it receives a shift request, it handles the shifting process.

Note that the dataflow diagram only handles the flow of information. It does not necessarily show transfer of control or sequencing.

Figure 1.

Architecture diagram


Speedometer monitor
Current speed
Gear selection
Clutch manager
Engine status
Change gears
Current speed
Clutch/shift/declutch
Engine status monitor
Engine status
Engine/clutch timing and control
Change RPM
Engine manager
Tasks
Inter-process messages

In this figure, the cruise control system is described as a set of asynchronous tasks. The paths describe interprocess messages between the tasks. You can think of these tasks as running simultaneously, synchronized by the intertask messages. This architecture seems to correspond nearly exactly to the dataflow diagram. At a more detailed level, however, it can become more complicated. Each task should have an accompanying document that describes how that task reacts to messages and what messages it sends out.

Figure 2.

a single person responsible for the project as a whole would reduce the problem of a fragmentary program in which the various parts are conceived independently and may or may not work together.

But the idea of separate architects and programmers must be tempered if it's to work at all. The designer and the programmer have to maintain communication; you need those feedback loops.

Q Who should be responsible for the overall design of the program?

A *Somebody* should be responsible for the overall design, but the designer will also depend upon constant feedback. In a large project, it is very difficult for an individual to handle all the technical detail. So eventually everyone ends up being involved. This continues as you get into the coding too. Things tend to show up, even if you've been careful.

I think this is where we get into the infamous design review, where everyone argues about the best way someone else should do something, and the manager with the most clout or the loudest voice gets his or her way.

The purpose of the design review is to set up the feedback loop so the designer or programmer can get the benefit of other people's experience. Many engineers do this informally when they talk to their coworkers about their projects. Unfortunately, these design reviews can be abused.

Q I notice that some of the more formal methodologies have very detailed rules concerning which managers should be present, or whether the original software author should attend at all. Is all that necessary?

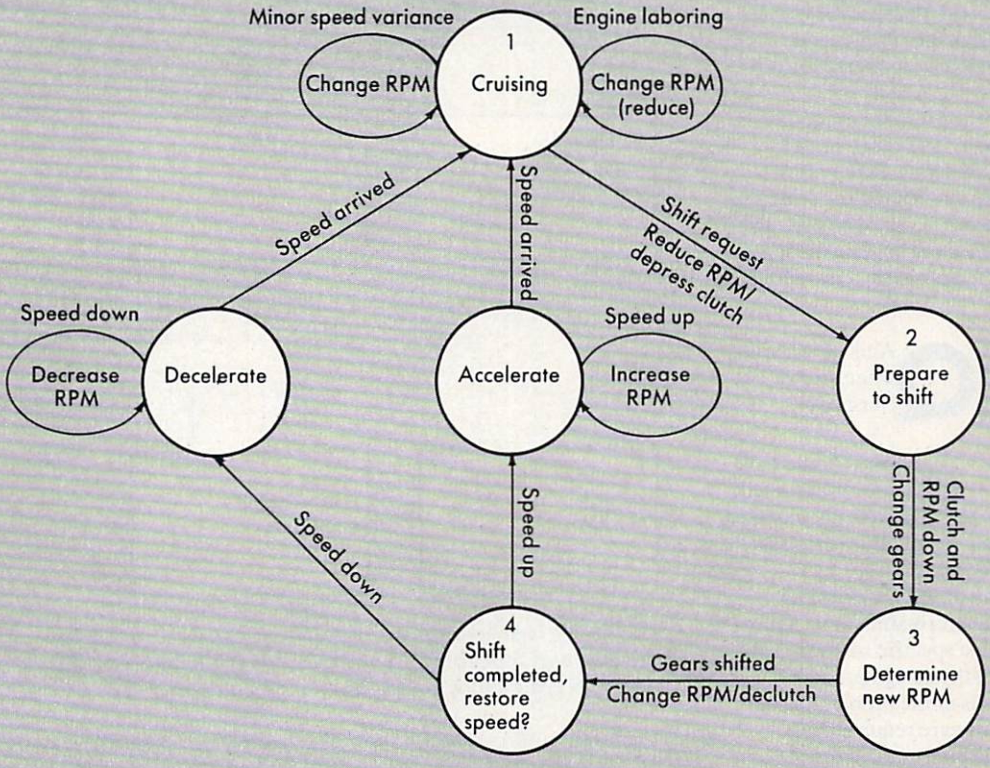
A Most of those kinds of rules exist to prevent the misuse of the review. But you can get carried away with the rules and totally miss the point of the review session. What you want to do is catch possible design errors early, preferably before you've started coding.

You've really got to use judgement with reviews. It isn't necessary to review everything that comes along. The work of a trusted, senior software engineer or the design of a simple module might not require a review. If it's not cost effective, then you shouldn't bother.

A lot of programmers make caustic comments concerning design-by-committee programming vs. software design done by one creative individual.

In the case of a small project, it's easy

State diagram


Minor speed variance
Engine laboring
1
Change RPM
Cruising
Change RPM (reduce)
Speed arrived
Speed arrived
Shift request
Reduce RPM/ depress clutch
Speed down
Decrease RPM
Decelerate
Accelerate
Speed up
Increase RPM
2
Prepare to shift
Speed down
Speed up
Clutch and RPM down
Change gears
4
Shift completed, restore speed?
Gears shifted
Change RPM/declutch
3
Determine new RPM

This state diagram shows the clutch/engine timing and control process (Process 2) of Figure 1. Each balloon represents a state and each path is a state change. The caption above a path is the trigger that causes the change. The caption below it is the action than accompanies the change.

For example, in state 1, cruising, the process is constantly monitoring the speed. If there is a variance from the requested speed, a control message is sent to the engine manager to change the RPM. In this case, the next state remains as state 1. However, if a shift request arrives from Process 1, then a state change occurs, and control messages are sent to the clutch and engine managers. This diagram has been greatly simplified; many triggers and states have been ignored.

Figure 3.

for one person to do most of the work. When things get bigger, though, it's difficult for one person to know everything about all aspects of the project. In industrial automation, you may be concerned about the properties of motors, position control, data bases, sequencing logic, and so on. And they've all got to work together. Even though one person needs to watch over the whole thing in general, you still have to get feedback from the various specialists.

Q Pseudocode and Structured English are often used when designing software. But pseudocode often looks something like Pascal with loose syntax. Why not just do it in real-code and save the bother of translating?

A I prefer to use Structured English over pseudocode since it's easier to think in English than in any procedural language. You use pseudocode or Structured English when you're laying out the structure of the program, usually at a high level. With Structured English, you can describe the program at an abstract level easily without being concerned with the specifics of the programming language. Pseudocode does the same thing at a lower, module or subroutine level.

If the logic of the program or module is obvious, then there isn't much need to spend time pseudocoding. After all, its purpose is to help you lay out the structure of a complex module before you code it. Or you can use it also to explain the logical structure of the program to someone else.

Q And what about when you're finished?

A You should have a well-designed, well-documented, efficient, and easy to maintain software product—more or less.

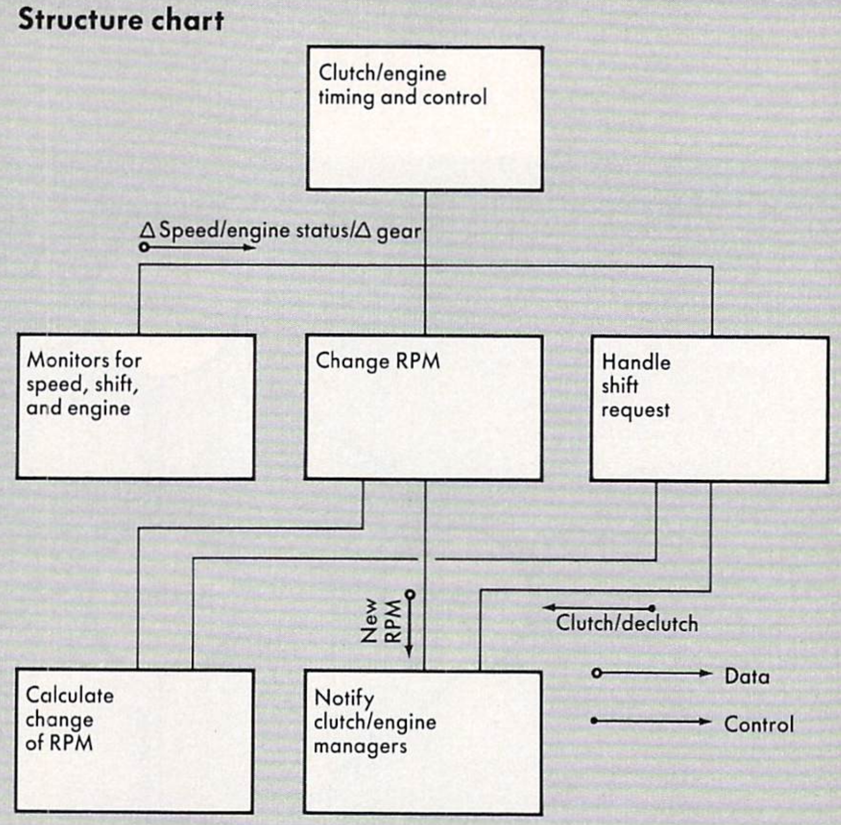
The philosophy of software design is to aid in the transformation of an amorphous problem to a specific software solution. The steps of the design process include an analysis of the problem, specification of the software requirements, then design of the program—from the overall structure on down to the design of the modules. The process consists of numerous iterations including reviews to make sure that you're doing what you intend to do, and that what you're doing will work.

How you approach design, what method you use, and how completely you adhere to any formal method is a matter of judgement. Sticking slavishly to a methodology excessively strict for a project that is small or simple can stunt the project altogether. Omitting the major steps of design on a larger or more complex project can lead to chaos.

Design reviews are useful, even if it simply means talking to another programmer or engineer about your project. On the other hand, a module that is relatively simple may not deserve so much attention.

Diagrams provide a pictorial description of the problem or of the general structure of the program. Structured English and pseudocode are used when laying out the logic of a program or module. Formality is not important—consistency and clarity are. After all, the final product is a program that works, works correctly, and does what it is supposed to do.

If you happen to be interested in software design, there are a couple of books you might want to look into. *Software Engineering: A Practitioner's Approach* by Roger S. Pressman (McGraw-Hill) gives a practical overview of methods and tools of software design. A more formal approach is found in *Structured Systems Analysis: Tools & Techniques* by Chris Gane and Trish Sarson (MCAUTO/IST). Also of interest is *Structured Design* by Edward Yourdon and L. Constantine (Prentice-Hall).


Structure chart
Clutch/engine timing and control
Δ Speed/engine status/Δ gear
Monitors for speed, shift, and engine
Change RPM
Handle shift request
New RPM
Clutch/declutch
Calculate change of RPM
Notify clutch/engine managers
Data
Control

This figure is an example of a structure chart, showing the relationships of the modules that compose the clutch/engine timing and control process. This diagram is a picture of the subroutine linkages within a larger routine.

Each subroutine or collection of subroutines is represented by a box. Each path represents a subroutine call. Only a few of the passed parameters have been drawn here, and some of the boxes represent several subroutines. Again, this was done for the sake of simplicity. A full scale structure chart can be very complicated.

Note that data and control parameters are distinguished by the open or closed dot on the flow arrows. Note also the shared subroutines at the bottom of the chart. A structure chart is useful for showing the linkages between routines in a single task environment, while the architecture diagram is useful in a multitasking environment.

Figure 4.

COMPUTERVISIONS

A personal visit with Donald Knuth

By Jan C. Shaw

The tall, gawky tuba player hastened across campus, avoiding the indignity of an all-out sprint. His efforts, however, were wasted. The band bus had already pulled out when he huffed into the parking lot behind the music building, his breath billowing clouds in the frozen Cleveland winter.

The Case Institute of Technology sophomore was conscience stricken at oversleeping but began to perk up at the thought of an entire day to do with as he pleased. And what he pleased on that winter morning in 1957, trudging back across campus to the old building with the computer rooms stuck in a corner, was to try to solve a problem that his math teacher said had never been solved in the centuries since it was posed. The teacher had promised an automatic A to any student who could answer the problem.

He found a room. Sat down. Concentrated. Figured. Concentrated. Figured. Blinked. Checked the figures. Yes. He'd done it.

He was 19 years old.

"I find a new solution to that problem every few years," Donald Knuth, Stanford University's resident computer science and mathematics genius, says today. "I mail them to my old professor. He didn't tell us at the time, but actually the problem *had* been solved once or twice before—only we didn't know that."

When Knuth solved that math problem he was a suffering physics major—"terrified," he says, of the required welding labs with their thousands of volts of electricity arcing about. "I was too tall for the equipment. My glasses wouldn't fit under the safety goggles so I couldn't see."

After solving the problem, he switched his major to math and began to spend more time with computers—against the advice of a few teachers who insisted computers were a dead-end business.

Today, at 46, Knuth is regarded as the world's top scholar in computer science. In addition to being an extraordinary mathematician, he is an accomplished musician, composer, teacher, and inventor. He was awarded the National Medal of Science four years ago—the nation's highest scientific honor—and it sits in his home office amidst a profusion of other awards. His articles have appeared in scores of scientific journals and computer magazines, although the one that makes him grin fondly is his first publication as a college freshman—in *MAD* magazine.

Knuth's monumental *The Art of Computer Programming*, a series of volumes that are computer science's standard of reference works, made his name 15 years ago. Computer scientists and mathematicians agree that Knuth's work introduced order, clarity, and depth to a young, fragmented discipline. He is regarded as the fountainhead of his field, its great pioneer—and he has completed only three of *The Art of Computer Programming*'s seven projected volumes.

Flashbulbs lend glamour to the tumult of the international typography conference at Stanford. The crowd's exuberance and the cheers are for Don Knuth. At the podium, his 6-foot-4 frame looking a bit rumpled, he lectures 150 typographers from North America, Europe, Japan, and India on his printing inventions.

Four years ago, Knuth strayed from his work on *The Art of Computer Programming* when the second edition of one of his books, computer-printed, left him aghast. "Ugly," he recalls, grimacing. The book offended all of his rather acute aesthetic senses. At first outraged and then curious, he delved into why the printing job was so unappealing. Solutions replaced questions. His subsequent applications of computer science and mathematics to the ancient arts of typeface design and typesetting revolutionized the printing world.

He set forth his inventions in two publications, *METAFONT* and *T$_E$X*. METAFONT is a system that uses classical mathematics to design alphabets. T$_E$X introduces a standard computer language for computer typography—a creation the importance of which has been compared with Gutenberg's invention of movable type. Knuth put both works in the public domain; neither he nor Stanford will collect a cent on them. "Proprietary stuff was holding back the field and it needed a push," Knuth says. "Besides, I just did it for the love of it."

"You know, a lot of people think the [computer] language is the important thing, but really, it's the way it's been implemented, the way it's been put on their computers that makes all the difference."

At the typography conference, Knuth leavens his discussion of the theory, background, and technical aspects of his work with some self-depreciating humor—an apology to the artists present who probably don't like math or computers even though Knuth finds great joy in bringing art to scientists and science to artists. There are wry chuckles from those artists. Pre-Knuth, these cognoscenti took years—sometimes decades—designing a new typeface. With his computerized system, they can design a new typeface in six months.

In the second row, sitting about eight feet from Knuth and listening attentively, is a white-haired, elegantly dressed German. His name is Hermann Zapf, and he is the world's leading typographer.

"Well," Zapf says in his precise, slightly accented English, "in a word, Knuth's a genius. It was he who introduced science into alphabet design. He invented a system which is, in my opinion, so flexible that it will bring back creative design to typography. I think METAFONT will become a tool in designers' hands to express our feeling and our thinking in the 20th century. I think METAFONT will get away from copying historical faces. I think Knuth's name will be recognized in the history of all the great masters."

Knuth doesn't think of it quite in those terms. Fame? Well he likes it. Remembered as a master? Well, he likes that, too, and keeps his notes for posterity. ("Historians need to know about your original mistakes," he says.)

But fame is not what motivates him. It is more basic than that: he enjoys his work and he feels obliged to do it. Take the book on Bible study he wants to write. "There is quite a need for scientists to say something about religion," he explains. Finishing *The Art of Computer Programming* series? He, well, he should finish the books sketched in his mind to do what God—in creating intelligence—designed him to do. Knuth's quiet Christianity is the foundation, the essence, of his life.

He actively participates in the Bethany Lutheran Church in Menlo Park, Calif.—organ recitals, choir committees. He takes the choir to the Dutch Goose, a local beer joint, after the practice. The church's pastor, the Rev. Bob Nicholus, doesn't mind that one of the baritones sometimes gets distracted during the third movement. Pastor Nicholus doesn't care that Knuth wears Birkenstock sandals with socks, instead of wing tips, to Sunday services. Pastor Nicholus would like to clone Don Knuth.

Knuth leans back in a comfortable, cushioned chair in his corner campus office, his worn-stockinged feet protruding from the side of the desk.

"There's a special kind of person called a computer scientist," he says looking up. "And they're going to be using computers at a higher level than most other people just because they have this special talent for it."

Knuth has commented that he sometimes can feel a shift within him between computer science and mathematics as he moves from one area to another and the spaces in between.

And computer scientists, he doubts, will probably only rarely use expert systems.

"But as they develop those systems, they will develop tools for computer science. But I doubt I would ever go to a computer and ask it to write a program for me. But the people who are developing programs that write programs are also developing techniques that I would use when I write programs myself. I don't use the products of the research but I use the ideas, the fruits of the research they generate."

He talks rapidly; his hands, incongruously graceful, move. Even when he is searching his way through a subject, he talks in final drafts. If he heard a startling remark, he wouldn't just blurt out, "That's the most incredible thing I've ever heard." He'd pause, think, and then say, "That's the third most incredible thing I've ever heard."

He sits forward.

"You know, a lot of people think the [computer] language is the important thing. But really, it's the way it's been implemented, the way it's been put on their computers that makes all the difference, not the fact that it's a good or bad language. What's important is how the language fits into other things on your computer."

Knuth's opinions are strong and definite when he has them. And he has very strong ideas on literate programming.

"The ideal traits of programmers in the coming years—well, the first thing is the ability to communicate with human beings. That means writing ability. Enthusiasm comes next. Then you have to have the ability to keep track of separate levels of abstraction. And a prerequisite to all of it is a good understanding of algorithms."

The ability to write. The ability to communicate with human beings. Familiar and important themes with Knuth.

To this pioneer, programs should be works of literature.

For starters, it's more fun, more exciting that way, he says. The programs are explained better and *are* better because writing literate programs encourages the programmer to do a better job.

He calls it literate programming and he recently wrote an article for a British computer journal with just that name.

In it, he writes that his ideas of literate programming have been embodied in a language and a suite of computer programs he developed over the last few years called WEB. It is chiefly a combination of two other languages: a document formatting language and a programming language. He uses $\TeX$ and Pascal but said that Ada, ALGOL, LISP, COBOL, FORTRAN, APL, C or even assembly language could be used.

"I believe that I have stumbled on a way of programming that produces better programs that are more portable and more easily understood and maintained," he writes. WEB grew out of his typography work. That pleases him because his typography work, which at first appeared to be a digression, ended by coming full circle to apply to "the heart of computer science."

But it's not for everybody, he says. It's for the "subset of computer scientists who like to write and to explain what they are doing."

Knuth wouldn't mind, someday, seeing a Pulitzer prize awarded for a computer program.

"My hope is that the ability to make explanations more natural will cause more programmers to discover the joys of literate programming because I believe it's quite a pleasure to combine verbal and mathematical skill."

"But," he says, "perhaps I'm hoping for too much."

But Knuth wouldn't mind, someday, seeing a Pulitzer prize awarded for a computer program.

Outside his window, the trees and red tiles of the roof reflect the light of the Monday morning.

Don Knuth doesn't like Mondays. Monday is mail day. It's also the day he reserves for saying "no" to requests to lecture, to teach, to write—many involving all-expenses-paid trips to exotic locales. He doesn't like saying no. And on this Monday morning, at his desk, wispy hair and pale eyes reflecting the light from the window, Knuth's mail is piled very, very high. Sometimes, he says, it takes the whole day just to separate the mail into "easy to answer" and "hard to answer."

Leaning back in his chair, Knuth does not look like a man driven to perfection. But what he's saying is another matter. He's talking about a favorite place of his—the sculpture-filled Frogner Park in Oslo, Norway.

"Gustav Vigelund was a sculptor. You go into the park, and it is filled with the fantastic monuments he sculpted. He started about 1910 and ended in the 40s when he died. And all his life he worked on monumental sculptures. What impresses me in the park is that here are these monuments in granite and bronze, more than 100, and they form a complete set. He finished his life's work."

When Knuth describes Frogner Park, his voice, in the stillness, has an edge of longing to it.

"So sometimes I feel a strong compulsion—that I have ideas inside of me that I want to get out. It is almost too strong a compulsion."

For years, day in and day out, Knuth made a list each morning of what to do that day and he accomplished everything on the list, no matter how late into the night he had to work. In 1967, as he worked on the second volume of *The Art of Computer Programming*, that regimen put him in the hospital with a bleeding ulcer. And although he still works continuously, the brush with serious illness changed the way he works.

"I resigned from all kinds of obligations. It was then I began to appreciate beauty. I started to read great books that weren't assigned to me. It just dawned on me—why shouldn't I be more human?"

His large, contemporary home on the Stanford campus is filled with art—original paintings, sculptures, and hangings, a few done by Don or his wife, Jill. The two of them designed the house, from her big art studio to his big book-and-file-lined study. They met at the Case Institute of Technology when she was a student at nearby Western Reserve University. They married in 1961 after she took her last two years at the Cleveland Institute of Art. They have two teen-age children, John and Jenny.

Knuth schedules his weeks ("no's" and mail on Monday, meetings on Tuesdays, privacy on Wednesdays, doctoral students on Thursdays, research on Fridays) for the express purpose of having time for his family. On his next sabbatical, to "make up a bit for all Jill has done," he will spend his time doing the housework, shopping, laundry, and cooking. It seems only fair to him.

Sitting at the dining room table, drinking a large glass of iced Mountain Dew, Knuth is reminiscing about *The Art of Computer Programming*. In 1962, when he was a young, newly married doctoral student at the California Institute of Technology, a publisher approached him to write a little computer book. He agreed, and began the project that was to make him famous.

"There was a lot being written about computers at the time," Knuth says, "but half of it was wrong and other things were in places you couldn't find. It was all very new and there weren't any accepted standards of quality. There wasn't even such a thing as computer science. What was published was so bad that no one bothered to read it."

"It was very important at the time in my own thinking to summarize everything that was known about computer programming," he says.

He succeeded beyond his wildest dreams. About 2,200 of the volumes are sold each month, for about $25 to $30 each. They have been translated into Russian, Japanese, Chinese, Romanian, and Spanish. A Portuguese translation is under way.

The resultant fame has some awkward aspects for Knuth. "It's harder to find friends who see me as an ordinary guy," he says. It was nice when I knew in my heart I was better than most thought I was. I could surprise them. Now it's the opposite. I'm worse than they think."

As he gets up to go to the music room he has to step over the family dog whom he eyes suspiciously. The little animal eyes him warily back. They have reached an accommodation: mutual distrust.

In the music room at the front of the house are two pianos (he and Jill play duets), a viola, a small antique pump organ, built-in shelves jammed with sheet music, and a custom-made pipe organ that Knuth designed.

Knuth almost went into music instead of science. Now, the music gives him his moments of contentment.

Knuth's pipe organ fills one end of the room. It is a massive instrument, towering 16 feet in the air. Barefoot, Knuth seats himself on the high, wide bench. Light cascading from an immense translucent window throws him into silhouette. In the silence he stills himself. And then he plays the opening notes of a Bach prelude in C minor. When he finishes, the last whispers of the music echo through the house.

Knuth looks up, smiling, content—the gawky tuba player no longer.

Jan C. Shaw is a reporter for the Business Journal *in San Jose, Calif.*

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES

- **Programmer's Referral List**
- **Compare Products**
- **Help find a Publisher**
- **Evaluation Literature free**
- **BULLETIN BOARD - 7 PM to 7 AM 617-826-4086**
- **Dealer's Inquire**
- **Newsletter**
- **Rush Order**
- **Over 300 products**

PUBLIC DOMAIN Research - Free

6 months paid research gives you leverage, learning. We found, combined, added to the best. All run, have source in C or ASM. Order $150 + get one free: Database, Editors, Modems, MSDOS RAMdisks & utils, Games in C.

RECENT DISCOVERIES

BRIEF Programmer's Editor is terrific for PCDOS. Worth effort to switch. Powerful, flows well. Macros, reconfigure. Contest $195

"C" LANGUAGE

	LIST PRICE	OUR PRICE
MSDOS: C86-8087, reliable	$395	call
Desmet with debugger	159	145
Lattice 2.1 - improved	500	call
Microsoft C 2.x	500	349
Williams - NEW, debugger	500	call
CPM80' Ecosoft C-now solid, full	250	225
BDS C - solid value	150	125
MACINTOSH: In stock!	NA	385
LINT-like for C86, Lattice	NA	400
Source-level Profiler: C86, Lat	NA	150
MACINTOSH: Full, ASM	NA	385

Compare, evaluate, consider other Cs

BASIC

	ENVIRONMENT		
BASCOM-86 - MicroSoft	8086	395	279
BASIC Dev't System	PCDOS	79	72
BASICA Compiler - BetterBASIC - 640K	PCDOS	—	199
CB-86 - DRI	CPM86	600	439
Prof. BASIC Compiler	PCDOS	345	325
MACINTOSH COMPILER with BASICA syntax	MAC	NA	325

Ask about ISAM, other addons for BASIC

FEATURES

Graphic C with full source - scientific plots, 4096 res. Optional 8087.C86, Desmet $195

PROLOG86 Interpreter for MSDOS includes tutorials, reference and good examples. Learn in first few hours. For Prototyping, Natural Language or AI. $125.

EDITORS Programming

BRIEF - Intuitive, flexible	PCDOS	NA	195
C Screen with source	8080/86	NA	75
FINAL WORD - for manuals	8080/86	300	215
MINCE - like EMACS	CPM, PCDOS	175	149
PMATE - powerful	CPM	195	175
	8086	225	195
VEDIT - full, liked	CPM, PCDOS	150	119
	8086	200	159

UNIX PC

COHERENT - for "C" users	PClike	$500	475
VENIX - "true V7" w/FTN	PClike	800	775
XENIX - "true S3" - rich	PC	1350	1285

Ask about run-times, applications, DOS compatibility, other alternatives UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

C to dBASE interface	8080/85	$150	$140
C Tools 1 - String, Screen	PCDOS	NA	115
C Tools 2 - OS Interface	PCDOS	NA	92
GRAPHICS: GSX - 80	CPM80	NA	75
HALO - fast, full	PCDOS	200	175
Greenleaf for C - full	PCDOS	NA	165
ISAM: C Index + -no royalties	MSDOS	NA	400
BTRIEVE - many languages	PCDOS	245	215
PHACT - with C	PCDOS	NA	250
PASCAL TOOLS - Blaise	PCDOS	NA	115
SCREEN: PANEL-86 - many languages	PCDOS	295	265
WINDOWS for C	PCDOS	NA	139

Ask about many others for FTN, BASIC, PASCAL, C-ISAM, Screen, Stat, Graphics.

PASCAL

	ENVIRONMENT	LIST PRICE	OUR PRICE
PASCAL MT + 86	CPM86/IBM	$400	$279
MS PASCAL 86	MSDOS	300	215
PASCAL 64 - nearly full	COM 64	99	89

OTHER PRODUCTS

AKA ALIAS - improve DOS	PCDOS	NA	60
Assembler & Tools - DRI	8086	200	159
CODESMITH-86 - debug	PCDOS	149	139
Disk Mechanic - rebuild	MSDOS	70	65
IQ LISP - full 1000K RAM	PCDOS	175	call
MBP Cobol-86 - fast	8086	750	695
MicroPROLOG	PCDOS	NA	265
Microsoft MASM-86	MSDOS	100	85
MS Fortran - improvements	MSDOS	350	255
Multilink-Multitasking	PCDOS	295	265
PL/1-86	8086	750	495
PLINK-86 - overlays	8086	350	315
Polylibrarian - thorough	MSDOS	99	89
PROFILER-86 - easier	MSDOS	NA	125
PROFILER - flexible	MSDOS	NA	175
Programmers Tlkt w/source	8086	NA	135
READ CPM86 from PCDOS	PCDOS	NA	55
READ PCDOS on an IBM PC	CPM86	NA	55
TRACE86 debugger ASM	MSDOS	125	115

Note: All prices subject to change without notice. Mention this ad. Some prices are specials.

Ask about COD and POs.
All formats available.

Call for a catalog, literature, and answers

800-421-8006

THE PROGRAMMER'S SHOP™
128-I Rockland Street, Hanover, MA 02339.
Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

CIRCLE 52 ON READER SERVICE CARD

PROLOG-86™

Learn Fast, Experiment

1 or 2 pages of PROLOG would require 10 or 15 pages in "C."

Be familiar in **one evening.** In a **few days** enhance artificial intelligence programs included like:

- **an Expert System**
- **Natural Language** (generates dBASE display)

Intro Price: $125 for PCDOS, CPM-86.
Full Refund if not satisfied.

CONTEST: "Artificial Intelligence Concepts"

$1,000 Prize, Recognition for applications in PROLOG-86™ that teach, are clear, illustrate. Call for details. Deadline 11/31/84

SOLUTION SYSTEMS™
45-D Accord Park, Norwell, MA 02061
617-871-5435

CIRCLE 60 ON READER SERVICE CARD

C Helper™

FIRST-AID FOR C PROGRAMS

Save time and frustration when analyzing and manipulating C programs. Use C HELPER's UNIX-like utilities which include:

DIFF and **CMP** – for "intelligent" file comparisons.
XREF – cross references variables by function and line.
C Flow Chart – shows what functions call each other.
C Beautifier – make source more regular and readable.
GREP – search for sophisticated patterns in text.

There are several other utilities that help with converting from one C compiler to another and with printing programs.

C Helper is written in portable C and includes both full source code and executable files for $135 for MS-DOS, CPM-80 or CPM-86. Use VISA, Master Card or COD.
Call: **617-659-1571**

Solution Systems™
335-L Washington Street
Norwell, MA 02061

CIRCLE 61 ON READER SERVICE CARD

An Implementation Demonstrating C Portability

How to implement non-standard data structures with symbol table utilities

By David Harry, Ph.D.

The rising importance of C as the standard development language for 16- and 32-bit systems is due largely to the ease with which programs may be moved from one processor to another. Software developers who are interested in quickly transporting software from one machine to another are advised to write their code in C and to write it in a portable fashion.

Therein lies the trick and the subject of this article.

Writing "portable" is easier said than done, especially when complex data structures are involved. I encounter this problem frequently, most recently while converting a set of FORTRAN B-tree utilities to UNIX-C.

I reflected upon how much time I had spent over the last 10 years converting code from one machine to another, browsed through my most recent copy of *Computer Design*, considered the bewildering array of new processors on the way, and resolved to write portable from now on.

Between resolution and implementation came the realization that I might have to give up some of those elegantly compressed data arrays containing many different types, all appearing rather randomly. No C compiler, or any other for that matter, can cope with multiple data types dynamically stored into common arrays in any random order and do so in a machine-independent way.

Consider an example to clarify the point:

```
funct1(cptr)
char *cptr;
{
float i=1.0;
*cptr+ + =’a’;
*cptr+ + =’\0’;
*((float *)cptr)=i;
cptr+ =4;
}
```

This function should compile without error but will it work? If it depends upon the processor, then this is decidedly not portable code. Consider what could happen on a Motorola 68000-based system. On entry to the function *funct*, the character pointer *cptr* could point to an even or odd byte boundary with equal probability. The code, as written, will execute without error if *cptr* contains an even address. However, if it is odd, it will not execute because the 68000 requires that all floats be stored only at even byte boundaries.

The complications continue. It is necessary to explicitly increment *cptr* four character storage units past the float rather than state *cptr++*. The dynamic interchange of data types that are not specifically understood by the compiler forces the programmer to do all the record keeping.

Given that potentially disastrous and non-portable consequences arise from such practices, why not just avoid them? Most applications where such data structures might be contemplated can be avoided by using a structured data type. Situations occur, however, where structured data types will not suffice, particularly when the occurrence of data types cannot be predefined.

This article describes a system of symbol table utilities based upon the digital searching algorithm. They serve as a good example of how to implement non-standard data structures in a portable fashion. Those not interested in the particulars of non-standard data structures may still use the symbol table utilities to their advantage since they represent a complete system of symbol table management that I have used in several compilers and compiler writing systems.

A table set up for digital searching may be thought of as an in-core B-tree. The symbols are entered into the table character by character. The first level of the digital searching tree contains all the unique first characters of all table entries, the second level contains all the unique second characters, and so on down the tree. Figure 1 illustrates a digital searching tree with a depth 3 and containing eight symbols having a maximum length of 3.

Each level contains a list of all possible alternate characters that can be found at the given position in the entered symbols. The first level contains all possible alternates found in the first position in the entered symbols, and so on. The horizontal arrows represent pointers between successive alternate choices. Vertical lines represent inter-level (successor) pointers, and the asterisks mark the end of a symbol.

To search such a table, the first character of a token is compared with the first entry in Level 1 of the table. If a match occurs, the successor pointer is followed down to the next level where the second character of the token is compared.

As long as successful comparisons occur, one continues descending in the table until the entire token is matched (symbol found) or the table terminates (failure). If a character comparison is false, the current alternate pointer is followed and the same token character is compared to the next alternate at the current level. Since the alternates are arranged in ascending order, unsuccessful testing of alternates may be terminated as soon as the current token character is less than the current alternate.

Several modifications can be made to the digital searching table to reduce its size and to improve its performance. One important simplification can be made by observing that after descending through the first few levels of the table, only a few choices remain to be tested (i.e., little benefit is derived by continuing the digital searching process out to the last character of each of its entries).

It is far more frugal to terminate the digital searching process after the majority of the entries have been eliminated and to perform a linear search of the few remaining choices. Figure 2 illustrates this concept. In this example, the digital searching table is truncated after a depth of three and the remaining fractional tokens are entered as a sequence of linked lists.

With structured data types it is possible to implement the digital searching method described quite efficiently. Available on the *COMPUTER LANGUAGE* Bulletin Board Service are two program listings which contain data definitions and various utilities that create and search in-core digital searching tables. I will refrain from a detailed explanation of their operation. (If the reader is interested in obtaining a copy of these two listings, they can be downloaded from the *COMPUTER LANGUAGE* BBS remote RCP/M computer at (415) 957-9370. The listings are placed on disk as PORTC1.LTG and PORTC2.LTG.)

Listing 1 (printed here and also available on the BBS as CPORT1.LTG) contains a short main program that exercises all major features of the utilities. With minimal study their use should be clear.

Consideration of Figure 1 confirms that very few of the nodes (character positions) in the digital searching tree are completely full (i.e., many of the pointers don't point to anything). Also, most of the symbol designators (*'s) are empty. In short, a lot of empty space is in the table. This is an obligatory consequence of having to provide for insertion of a new symbol into the table at any point.

There are certain applications, however, where all symbols are entered into the table in advance and searched repeatedly thereafter. An example would be a reserved word table for a compiler. In such cases it is possible to compress the digital searching table into a much more compact form, eliminating the unnecessary pointers and empty symbol designators.

Specifically, it is possible to eliminate successor pointers entirely. By including a status flag in each node, it is also possible to selectively exclude alternate pointers and symbol designators. A further compression of the table may be performed by converting all remaining pointers to indexes relative to a base address of zero and define them as data type *short unsigned*. This allows compressed tables of a maximum size 64K on the 68000, which is more than adequate for any practical reserved word table.

An additional benefit of converting to indexes rather than pointers is that the table is independent of its address in memory and may even be written to external stores and subsequently used within other program modules.

As a consequence of all this ambitious byte pruning, several things have occurred. First, the possibility of continuing to express each table node as a structured data type has been eliminated. Second, the order in which individual data elements of differing types will appear in memory is no longer fixed. Thus, the

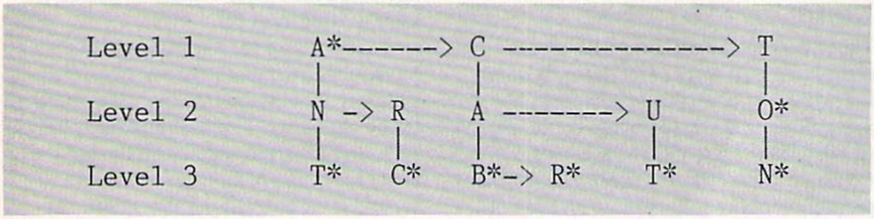


Figure 1.

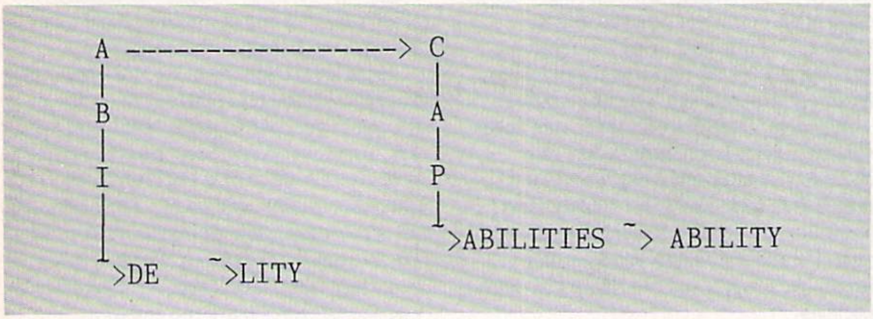


Figure 2.

```
LEVEL 1

NODE 1    (FLAG - "A" - SYMBOL # - ALT. INDEX)
NODE 2    (FLAG - "C" - ALT. INDEX)
NODE 3    (FLAG - "T" - ALT. INDEX - "END OF LEVEL 1")
```

Figure 3.

problems of memory alignment and nonportability have crept in. Figure 3 may better illustrate this point.

The structure of each node in the first level of a compressed digital searching tree, based upon the table illustrated in Figure 1, is schematically depicted. Observe that no empty data elements are present in any node. Also note that there is no standard order of occurrence of data elements within the nodes. Since each subsequent data element may have a different data type (*int*, *char*, *short*) than its predecessors, the problem with memory alignment and subsequent portability should be clear.

Any routines that build such tables must contain provisions for aligning the next data element properly, subject to the constraints of the target processor. Furthermore, the routine that searches tables must also contain provisions for anticipating the presence of filler space that has been inserted to maintain memory alignment and to appropriately index pointers around it. Normally this happens invisibly when standard data structures are employed. Use of compressed data structures has created the need for detailed coding of provisions. Those interested in C trivia may wish to investigate why a statement such as: _*ilgn(u)++=5;_ is illegal and must be written as stated.

In the case of the 68000, the program must insure that the appropriate pointers are adjusted to an even byte boundary whenever the data type changes from *char* to any other. Other processors have more restrictive alignment requirements than the 68000, so it is certain that a routine specifically coded for that processor may fail to execute properly on many others.

The concomitant problems of alignment and portability may be solved by use of a simple mechanism that takes advantage of C's lack of contraints regarding the use of "union" data types. As Brian Kernigan and Dennis Ritchie's book *The C Programming Language* states, "It is the responsibility of the programmer to keep track of what type is currently stored in a union; the results are machine dependent if something is stored as one type and extracted as another."[1] To see how this provision may be used to an advantage in solving the alignment/portability problem, consider the following short routine which contiguously stores three identical data blocks—each composed of a single character, an integer, and a float—into a character array.

The routine in Listing 2 will compile without error and should also run on any processor regardless of alignment restrictions. One should now refer to Listing 3 (printed here and available on the BBS as disk file CPORT3.LTG), which contains *algnm.h*. This demonstrates what an *align* data type is and also reveals that *ilgn()* and *flgn()* are simply macro expansions that operate on *align* data types. Observe that *align* is type defined as a union capable of containing a pointer to every legal C primitive data type.

In the routine in Listing 3 it was possible to insert differing data types into the character array in any order simply by changing the current value of "u" from one pointer type to another (just as the C specification declares that we may) subject to one crucial restriction. It becomes the programmer's responsibility to insure that the alignment is correct for the next data type. The macro expansions given in *algnm.h* perform this function for every legal data type. When one switches from one data type to another, the appropriate macro is invoked, which insures alignment is always maintained. Routines such as the previous one will correctly compile and execute on any processor for which compatible *algnm.h* macros (or functions) have been provided. For most processors, production of the seven necessary macro definitions should be straightforward.

It is now possible to return to the problem of creating compressed reserved word tables from digital searching trees and to consider routines that use this data alignment technique in a less trivial fashion. On the *COMPUTER LANGUAGE* BBS I've provided a program listing called PORTC3.LTG. This listing provides a set of routines that create and search a reserved word table using the previous *algnm.h* memory alignment macros. They accept (as input) digital searching trees that are created by the routines in the PORTC2.LTG disk file on the *COMPUTER LANGUAGE* BBS and create (as output) a reserved word table that is compressed in all respects as described previously. The table is indexed and therefore memory-address independent, so it may be saved externally and subsequently read (or permanently loaded) into other programs.

The searching routine contains two compilation options. If *EXACT* is true, only exact matches will be found; if false, the longest partial matches found will also be reported. This searching routine is quite efficient and will identify tokens with an efficiency approaching that of a truly deterministic finite automation, such as the UNIX-based lex.

Several caveats are in order concerning the use of the *algnm.h* macros. If nonstandard data structures are moved in memory or stored on an external device and subsequently re-read, the base address of the data array must have the same memory alignment in all cases. For example, if the data structure base address falls on a word boundary when created, then it must always be constrained to reside at a word boundary. It is possible to create some truly spectacular errors using *algnm.h*, errors that the C compiler would never allow to propagate into run-time code. For example, a misplaced or missing alignment macro may cause a memory fetch instruction with an improperly aligned operand address. And this may result in some strange error message. On my DUAL Systems 83/20 I get a cryptic "bus error!" message.

Finally, I must admit that the digital searching routines presented here do not implement the *algnm.h* macros in the most bullet-proof fashion. This is due to the implicit assumption that character data types require no special alignment. Although this is true for most processors, it is by no means true for all. The Honeywell 6000, for example, has a different format for character pointers than for other data types. To be completely safe, one should invoke the *clgn()* macro when switching from any other data type to type *char*.

Reference

1. Kernigan, Brian and Ritchie, Dennis. 1978. *The C Programming Language*. Prentice-Hall, Inc. Inglewood Cliffs, N.J.

David Harry has a Ph.D. in physical biochemistry and is director of computer support services at the Univ. of Texas Medical Branch in Galveston, Texas.

```
/* dgt2.c - test the dgsm.c  and dgsx.c routines */

#include <stdio.h>
#include "dgsdefs.h"

#define NULL    0
#define MAXLEN  50
#define DEPTH   5

main()
{
FILE *fopen(),*fp;
struct tblctrl *tctrl;
struct xtbctrl *xctrl;
char fname[30],str[MAXLEN],*malloc();
int nsym;

for(;;)
{
fprintf(stdout,"Enter file name:  ");
if(fscanf(stdin,"%s",fname) != 1)
fprintf(stderr,"Syntax error on file name entry!");
else if((fp=fopen(fname,"r")) != NULL)
break;
else
fprintf(stderr,"Invalid file name!\n");
}
/* Initialize the control table */
tctrl=galloc();
intdgs(tctrl,DEPTH);
/* Read the tokens into the dgs */
nsym=0;
while(gtstr(fp,str,MAXLEN) != EOF)
{
fprintf(stdout,"%s\n",str);
nsym++;
if(dgs(tctrl,str,nsym) != nsym)
{
fprintf(stderr,"Error on # %3d, %s\n",nsym,str);
exit(1);
}
}
/* "Rewind" the input file      */
fclose(fp);
if((fp=fopen(fname,"r")) == NULL)
{
fprintf(stderr,"Unable to reopen input file!\n");
exit(1);
}
```

Listing 1.

Your CP/M-80 becomes THE definitive 8-bit Operating System

ZCPR3 SETS YOU FREE!

- Command Line IF-ELSE-GOTO, Job Flow Control Automation
- File, Disk, Memory, and System Status Utilities, all online
- Named Directories, each with optional password access
- Dynamically loadable Resident Command Packages
- Error Handling Packages, graceful recoveries
- Screen-oriented Menu Generators
- Complete online HELP system
- SHELLS, with shell variables
- File Search Paths, organize your system
- Multiple Commands per Line, program chaining
- Aliases, eliminates complex keystroke entries, nestable
- Shell generator, make any application program a SHELL
- Terminal Install utility, uses UC Berkeley TERMCAP database

ZCPR3 permits highest productivity while providing flexible system control and maximum usability. It's terrific! The utilities create a tool-set environment that's hard to beat even by large main-frame machines. You are free to do what you want in the way you want. Try it and see if you don't agree. Newsletter, User Group and 24-hour bulletin board report ZCPR3 community activities.

VISA/MASTERCARD, check or money order accepted. Specify disk format desired. Add $3.00 shipping and handling. Calif. add 6.5% sales tax. Phone or send order now to Echelon, Inc. — your single source for ZCPR3 related software, support and documentation. We market programs taking full advantage of ZCPR3 capabilities; send us yours for evaluation and report.

1. **CORE CP/M CCP REPLACEMENT MODULE**
   Starter-kit utilities on 2 disks, plus 173-page SAMPLER documentation .... only $39.00
2. **ZCPR3 UTILITIES**
   Complete, full source on 8 disks $89.00
3. **Z3-DOT-COM**
   Auto-install ZCPR3. Load-and-go, complete full-up ready-to-run system on 4 disks $149.00
4. **ZCPR3: THE MANUAL**
   Lavish, typeset, over 300 pages $19.95
5. **SYSLIB3 MACRO LIBRARY**
   Used to write most ZCPR3 utilities, documention and full source on 4 disks $29.00
6. **DISCAT**
   Menu-driven disk and file catalog system under ZCPR3 ... $49.00

Trademarks:
CP/M, Digital Research
ZCPR3, SYSLIB3, Richard L. Conn
Z3-Dot-Com, Alpha Systems Corp.
DISCAT, Echelon, Inc.

Echelon, Inc. 101 First Street • Los Altos, CA 94022 • 415/948-3820

CIRCLE 21 ON READER SERVICE CARD

Six Times Faster!

Super Fast Z80 Assembly Language Development Package

Z80ASM

- Complete Zilog Mnemonic set
- Full Macro facility
- Plain English error messages
- One or two pass operation
- Over 6000 lines/minute
- Supports nested INCLUDE files
- Allows external bytes, words, and expressions (EXT1 * EXT2)
- Labels significant to 16 characters even on externals (SLR Format Only)
- Integral cross-reference
- Upper/lower case optionally significant
- Conditional assembly
- Assemble code for execution at another address (PHASE & DEPHASE)
- Generates COM, HEX, or REL files
- COM files may start at other than 100H
- REL files may be in Microsoft format or SLR format
- Separate PROG, DATA & COMMON address spaces
- Accepts symbol definitions from the console
- Flexible listing facility includes TIME and DATE in listing (CP/M Plus Only)

SLRNK

- Links any combination of SLR format and Microsoft format REL files
- One or two pass operation allows output files up to 64K
- Generates HEX or COM files
- User may specify PROG, DATA, and COMMON loading addresses
- COM may start at other than 100H
- HEX files do not fill empty address space.
- Generate inter-module cross-reference and load map
- Save symbol table to disk in REL format for use in overlay generation
- Declare entry points from console
- The FASTEST Microsoft Compatible Linker available

SPEED! SPEED! SPEED!

- Complete Package Includes: Z80ASM, SLRNK, SLRIB - Librarian and Manual for just $199.99. Manual only, $30.
- Most formats available for Z80 CP/M, CDOS, & TURBODOS
- Terms: add $3 shipping US, others $7. PA add 6% sales tax

For more information or to order, call:

1-800-833-3061

In PA, (412) 282-0864

Or write: SLR SYSTEMS
1622 North Main Street, Butler, Pennsylvania 16001

CIRCLE 59 ON READER SERVICE CARD

SLR Systems

COMMODORE 64 GETS AWAY FROM BASICs with

THE Pascal Compiler™ FOR THE COMMODORE 64

Limbic Systems, Inc. introduces the PASCAL COMPILER from Oxford Computer Systems (Software), Ltd., developers of *PETSPEED -- the BASIC compiler recommended by Commodore.

THE PASCAL COMPILER OFFERS --

- **EASE OF OPERATION**
  Pascal, a programming language, simplifies development of software for customized applications
- **POWER**
  to write programs efficiently and effectively
- **TEACHING TOOL**
  Pascal is the preferred language to teach programming skills
- **EFFICIENCY**
  debugging time is minimized -- the major task in program development
- **FLEXIBILITY**
  utilities are provided as an aid in programming development

THE PASCAL COMPILER IS REASONABLY PRICED!

Limbic Systems Inc.
560 San Antonio Road, Suite 202 Palo Alto, CA 94306
(415) 424-0168

Commodore 64 is a trademark of Commodore Electronics, Inc.
**PETSPEED is a trademark of Oxford Computer Systems (Software), Ltd.*

CIRCLE 38 ON READER SERVICE CARD

```
nsym=0;
while(gtstr(fp,str,MAXLEN) != EOF)
{
nsym++;
if(srdgs(tctrl,str) != nsym)
fprintf(stdout,"Symbol #%3d not found!\n",nsym);
}
fclose(fp);
fprintf(stdout,"# entered: %3d\n",tctrl->nentry);
fprintf(stdout,"table size in bytes: %d\n",mtrsiz(tctrl));
fprintf(stdout,"# nodes: %3d  # links: %3d  #chars: %3d\n",
        tctrl->nnodes,tctrl->nlinks,tctrl->nchars);
xctrl=xalloc();
if(!cdgx(tctrl,xctrl))
{
fprintf(stderr,"Error in cdgx module!\n");
exit(1);
}
/* print the compressed tree nodes
xprt(xctrl);
*/
if((fp=fopen(fname,"r")) == NULL)
{
fprintf(stderr,"Unable to reopen input file!\n");
exit(1);
```

Listing 1 (Continued).

NEW Ver. 2.2
Easier — More Power

WINDOWS FOR C™

FOR THE IBM PC + COMPATIBLES
Lattice C, CI-C86, MWC86
DeSmet C, Microsoft C

C ADVANCED SCREEN MANAGEMENT MADE EASY

ADVANCED FEATURES

- Unlimited windows and text files
- Word wrap, auto scroll
- Horizontal and vertical scroll
- Fast! + No flicker or snow
- No memory in screen buffers
- Complete color control
- Auto memory management
- Save and move window images
- Easy overlay and restore
- Format and print with windows
- Highlighting

WINDOWS++

Much more than a window display system, **Windows for C** is a video display toolkit that simplifies all screen management tasks.

SIMPLIFY • IMPROVE

- Menus
- Data screens
- Form printing
- Help files
- Editors
- Games

ALL DISPLAYS

C SOURCE MODULES FOR
pop-up menus, multiple window displays, label printer, cursor control, text mode bar graphs.
plus complete building block subroutines

DESIGNED FOR PORTABILITY

FULL SOURCE AVAILABLE
NO ROYALTIES

WINDOWS FOR C	$150
(specify compiler & version)	
Demo disk and manual	$ 30
(applies toward purchase)	
Dealer Inquires welcome	

A PROFESSIONAL SOFTWARE TOOL FROM
CREATIVE SOLUTIONS
21 Elm Ave, Box L10, Richford, VT 05476

802-848-7738
Master Card & Visa Accepted
Shipping $2.50
VT residents add 4% tax

CIRCLE 15 ON READER SERVICE CARD

```
}
nsym=0;
while(gtstr(fp,str,MAXLEN) != EOF)
{
nsym++;
if(sgx(xctrl,str) != nsym)
fprintf(stdout,"Symbol #%3d not found in compressed tree!\n",nsym);
}
fclose(fp);
fretre(tctrl);
fprintf(stdout,"Size of compressed tree: %d\n",xctrl->size);
}

/*--- GTSTR ----------------------------------------------------
Return a non-null line from the requested input device.  Any line longer
than 'maxlength-1' is truncated.  The length of the line is normally
returned. EOF is returned on eof or error.
-------------------------------------------------------------------------*/

gtstr(fp,str,maxlength)
char *str;
int maxlength;
FILE *fp;

{
int c=0,i=0;
while(i==0 && c!=EOF)
{
while((c=getc(fp))!=EOF && c!='\n' && i<maxlength-1)
{
*str++=c;
i++;
}
}
*str='\0';
return(c==EOF ? EOF : i);
}
```

Listing 1 (Continued).

```
#include "algnm.h"
pfunct(cary,one_char,int_num,float_num)
char one_char, *cary;
int  int_num;
float float_num;
{
int i;
ALIGN u;  /* u is a union of pointers  */
u.cptr=cary;  /* refer to it as a char pointer */
for(i=0; i<3; i++) {
*u.cptr++=one_char;
*ilgn(u)=int_num;  /* convert u to an integer pointer */
u.iptr++;
*flgn(u)=float_num;  /* convert u to a float pointer */
u.fptr++;
}}
```

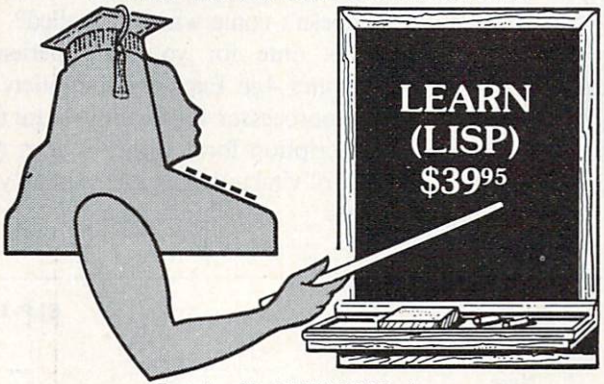
Listing 2.

```
/* algnm.h - vers. M68000-1.0
Memory alignment routines.  NOTE: These def's are highly machine-dependent.
*/

typedef union lgnm
{
int *iptr;
short *sptr;
float *fptr;
double *dptr;
char *cptr;
unsigned int *uptr;
unsigned short *usptr;
} ALIGN;

#define _lg(p)  (((int)p.cptr) & 1 ? ++p.cptr : p.cptr)
#define clgn(p) p.cptr
#define ilgn(p) ((int*)_lg(p))
#define slgn(p) ((short*)_lg(p))
#define flgn(p) ((float*)_lg(p))
#define dlgn(p) ((double*)_lg(p))
#define ulgn(p) ((unsigned*)_lg(p))
#define uslgn(p)        ((unsigned short*)_lg(p))
```

Listing 3.

(LISP)

UO-LISP Programming Environment
The Powerful Implementation
of LISP for MICRO COMPUTERS



Featuring: The LEARN LISP Package

Complete with LISP Tutorial Guide, Editor Tutorial Guide, System Manual with Usage Examples, Full LISP Interpreter, On-Line Help and other Utilities. LEARN LISP fundamentals and programming techniques rapidly and effectively for $39.95.

UO-LISP Production Power

The Usual LISP Interpreter Functions, Data Types, Structure Editor, Screen Editor with Mouse Support option, Compiler and LAP Assembler, Optimizer, LISP & Assembly Code Intermixing, Compiled Code Library Loader, Numerous Utility Packages, Hardware and Operating System Access, Session Freeze and Restart, Comprehensive 350 Page Manual with Usage Examples, and much more is available in the UO-LISP Programming Environment. Configurations start at $100.00, Manual may be purchased separately - $40.00.

Other UO-LISP products include **Little META** a translator writing system, **RLISP** a high level language, and **LISPTEX,** a text formatter.

REQUIRES: The UO-LISP Programming Environment runs on most Z80 computers with CP/M or TRSDOS. The 8086 version available soon.

Visa and Mastercard accepted.

Northwest Computer Algorithms
P.O. Box 90995, Long Beach, CA 90809
Phone Orders Accepted - (213) 426-1893

CIRCLE 46 ON READER SERVICE CARD

SMALL C FOR IBM-PC

Small-C Compiler Version 2.1 for PC-DOS/MS-DOS
Source Code included for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library

$40

CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change variables all on the source level
Source code included

$40

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

ASM or MASM is required with compiler.
Include disk size (160k/320k), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation

CIRCLE 19 ON READER SERVICE CARD

ROBOTICS AGE

A Real-Time Experience

Learning how to program was the easy part. Now, what are you going to do with your knowledge? How about writing a check balancing program? Maybe you could start developing that wonderful data base to keep track of everyone's birthdays. However, there are more interesting challenges.

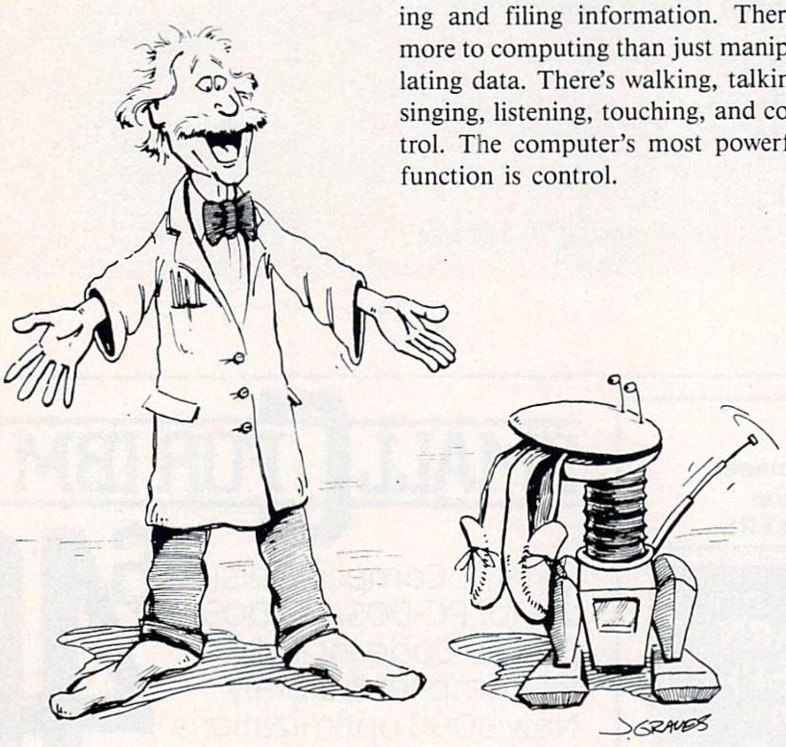
Computers are very useful for tracking and filing information. There's more to computing than just manipulating data. There's walking, talking, singing, listening, touching, and control. The computer's most powerful function is control.

Real-time computer techniques can control factories and machinery, monitor your home environment and protect it from intruders, and operate little mechanical friends which will take out the garbage.

The near future will show us machines which respond to human voice controls, are capable of finding their own way around a house or factory floor, and are able to make their own decisions. *Robotics Age* teaches you to design and work with the practical real-time applications of state-of-the-art microcomputer technology. Robots are simply machines which respond to their environments and can act on their own. Computers make these machines possible.

After all, many people claim their computers are user friendly—but how can your computer be user friendly if it doesn't come when it's called?

It's time for you to experience *Robotics Age*. Explore the frontiers of microprocessor applications. Use the subscription form below to start the flow of vital technical information you need.



CLP-1

YES! ***Sign me up TODAY for my personal subscription to Robotics Age, The Journal of Intelligent Machines.***

US Subscriptions	
☐ 12 issues	$24
☐ 24 issues	$45
☐ 36 issues	$63
Canada & Mexico	
☐ 12 issues	$28
☐ 24 issues	$53
☐ 36 issues	$75
Foreign	
☐ 12 issues (surface)	$32
☐ 12 issues (Air Mail)	$68
☐ 24 issues (surface)	$61
☐ 24 issues (Air Mail)	$133
☐ 36 issues (surface)	$87
☐ 36 issues (Air Mail)	$195

Non US Subscription Rates:
Payable in US funds, drawn on a US bank. Subscription length will be adjusted downward on a pro-rata basis for any currency conversion charges. Foreign subscription orders may be paid in US dollars via MasterCard or VISA.

MasterCard VISA

RETURN WITH PAYMENT TO:
Robotics Age, Box 358
Peterborough, NH 03458

Name
Company
Address
Town
State Zip/Postal Code Country

☐ Bill Me

Credit Card Information
☐ MasterCard
Card Number
☐ VISA
Expiration Date

Signature
Total amount Enclosed or Charged $ ________

CIRCLE 57 ON READER SERVICE CARD

Batch: A powerful IBM "Language"

By Darryl E. Rubin

The IBM PC-DOS manual is a marvel of understatement. It takes more than one reading before you really realize that PC-DOS Version 2 puts at your command a simple but extremely useful computer language called Batch.

Batch? A language? Indeed. With a little ingenuity, you can make this IBM utility play tricks you thought only Pascal or PL/I could provide—procedure calling, recursion, argument passing, string manipulation, looping, case selection, even file and console I/O. And even better, Batch is an extensible language. You can add to it as you go along.

Let's look at just a few of the surprising capabilities of Batch. I will also explain a powerful but undocumented feature of the language and give you some handy Batch programs that you can put to immediate use. To save some finger work, you can download all these programs from the *COMPUTER LANGUAGE* Bulletin Board Service by calling (415) 957-9370 and looking for the files marked BATCH.LTG. *Note: Examples in text use an indent to show that text should be on a single line.*

Let your files do the typing

To begin with, what qualifies something to be a programming language? Most books on programming languages will tell you that every language is founded on three primitive constructs: sequence, selection, and repetition. These happen to be the three primitive constructs of Batch.

The sequence construct lets you specify a set of statements to be executed one after another. In Pascal you do this by surrounding the statements with the reserved words *BEGIN* and *END*. In Batch you simply put the statements in a file and specify the file with the extension .BAT. Typing the name of a Batch file as a command causes PC-DOS to execute each statement in the file.

Batch statements include every PC-DOS command and program name plus some special commands that are unique only to the Batch language. For example, writing this article was a lot easier by creating a Batch file called WRITE.BAT:

```
EDIT ARTICLE.TXT
SPELL ARTICLE.TXT
FORMAT ARTICLE.TXT
PRINT ARTICLE.DOC
```

Listing 1 shows another, more sophisticated Batch file called BUILD.BAT. This one will turn an assembly language source file into an executable .COM file and erase the intermediate .OBJ and .EXE files. To use it, just type *BUILD PROGNAME*, where *PROGNAME* is the name of the source file (without the extension). Note, however, that your source file must conform to the PC-DOS convention for .COM programs.

BUILD also demonstrates two other Batch features: argument passing and string concatenation.

To perform argument passing, Batch assigns each word in your command line an argument name in the range %0 through %9. When you type *BUILD PROGNAME*, %0 is set to *BUILD* and %1 to *PROGNAME*.

Wherever an argument name appears in a Batch file, Batch substitutes it with that argument's value and concatenates the substituted value to any neighboring strings. For example, typing *BUILD PROGNAME* causes line 1 of BUILD.BAT to become *MASM PROGNAME:* and line 4 to become *DEL* PROGNAME.OBJ.

```
masm %1;
link %1;
exe2bin %1
del %1.obj
del %1.exe
del %1.com
ren %1.bin %1.com
```

Listing 1.

For a more complete demonstration, let's use one of Batch's special commands called *ECHO*. *ECHO* simply writes everything following it on the command line to standard output (except if you say *ECHO ON* or *ECHO OFF*— these have special meanings). You can use *ECHO* to put argument passing and concatenation through their paces (see Figure 1).

You might ask, what's the use of the %0 argument if its value is always the name of the Batch file being executed? We'll see shortly.

Constructs of choice

The second primitive language construct is selection. Selection lets you specify a choice between sets of statements to execute. Pascal provides the *IF* and *CASE* statements for this purpose.

Batch also has an *IF* statement that uses the simple format, *IF <condition>* *<statement>*. Very streamlined. For conditions, your choices are:

- String comparison:
IF <string1> == <string2>
- File existence:
IF EXIST <filename>
- *ERRORLEVEL* checking:
IF ERRORLEVEL n
(0 <= *n* <= 255)
- Optional *NOT*:
IF NOT <condition>

ERRORLEVEL is a PC-DOS system variable that can be set by any program. Saying *IF ERRORLEVEL n* really means *IF ERRORLEVEL <= n*. IBM meant for this to be used in programs that return error status to the PC-DOS shell, but more creative uses abound, as we'll see.

First though, let's play with *IF*. What does the following Batch file do?

```
MYSTERY.BAT:
  CD %0
  IF EXIST AUTOEXEC.BAT
     AUTOEXEC
```

The file connects you to a subdirectory called *MYSTERY* and executes a file called AUTOEXEC.BAT if it exists there.

Notice our use of the %0 parameter. If you named the file WORKDIR.BAT instead of MYSTERY.BAT, it would connect you to a subdirectory called *WORKDIR*. Name it WP.BAT or TOOLS.BAT or ANYTHING.BAT and it will connect you to those subdirectories. As you can see, *CD %0* is one versatile command.

To see another side of the Batch *IF* statement, look at the ATTR.BAT file in Listing 2. This command lets you play with video attributes. For example, *ATTR REVERSE* puts the screen into reverse video mode, *ATTR BOLD* puts it into highlighted mode, and *ATTR NORMAL* goes back to normal mode. You can even combine certain attributes. Try *ATTR BLINK USCORE* to see what blinking underscored characters look like!

ATTR.BAT introduces two other Batch commands—*SHIFT* and *GOTO*.

Shifty arguments

SHIFT makes each argument equal to its successor, which means it performs the sequence of assignments %0 := %1, %1 := %2, %2 := %3, etc. It has two uses: accessing command line arguments beyond the tenth one and processing a variable number of arguments. Lines 8-9 of ATTR.BAT demonstrate the second of these uses. We shift the arguments and test whether %1 becomes null. If not, we *GOTO* the start of the Batch file to process the next attribute.

Pay attention to the tricky way we test for null arguments. It doesn't work to do the obvious *IF %1 =* . (Do you know why?) However, saying *IF %1/ == /* works because when %1 is null we end up with *IF / == /*.

IF is not the only way to do selection in Batch. *GOTO* offers an alternative akin to the *CASE* statement of Pascal. For example, *GOTO %1* branches to a label whose value was passed as an argument. My first version of ATTR.BAT used this construct for jumping to the labels :NORMAL, :BOLD, :REVERSE, and so on.

Taken for a loop

I sneaked one in on you. Did you notice that ATTR.BAT also demonstrates repetition—the third primitive construct of programming languages? Yes, the good old *GOTO* plus the correct use of the *IF* statement is one way to execute a set of statements several times.

Another way is to use the *FOR* statement. Pascal has one and so does Batch. It looks like this:

```
FOR %%V IN (<list>) DO
     <statement>
```

Rather than get wordy, let's try some examples. The statement,

```
FOR %%E IN (Testing 1 2.) DO
   ECHO %%E
```

is equivalent to the sequence

```
ECHO Testing
ECHO 1
ECHO 2.
```

And the statement,

```
FOR %%F IN (JUNKFIL *.BAT) DO
     ERASE %%F
```

is the same as, say,

```
ERASE JUNKFIL
ERASE BUILD.BAT
ERASE ATTR.BAT
```

You can do some imaginative things with *FOR* loops. The following will verify that the argument %1 has one of the values 1, 2, or 3:

```
FOR %%A IN (1 2 3) DO
  IF %1 == %%A GOTO OK
ECHO %1 is not a valid value.
GOTO EXIT
:OK
```

And this line will print five copies of a file named by %1:

```
FOR %%N IN (1 2 3 4 5) DO
  PRINT %1
```

```
PARROT.BAT:   ECHO %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
              ECHO %0: %1%2 %3 th%4%t *%5%6* %7%8%9!

Arguments:      %0   %1  %2  %3  %4    %5 %6 %7 %8 %9
You type:   A>PARROT Th  is  is   a    te st
You see:      PARROT Th  is  is   a    te st
              PARROT: This is that *test* !
```

Table 1.

Teaming up with PC-DOS

Useful stuff but we're just beginning to warm up to Batch's most powerful capabilities. Surprisingly, the most powerful stuff arises not from features of Batch itself but from the combination of Batch's primitive constructs with PC-DOS's facilities for I/O redirection, filters, nested shells, and the string environment.

Let's start with I/O redirection. Program output under PC-DOS defaults to the display, but you can redirect it to another device or file by putting >*outname* on the command line, where outname is the desired device or file.

Remember the *ECHO* command? Try this:

```
REDIRECT.BAT: ECHO This is a
    test>junkfil
ECHO This is a test>lpt1:
```

This Batch file actually creates another file (junkfil) and writes to the printer too. Using *ECHO* like this, you could make a Batch file like ATTR.BAT which sets your printer to a desired font. Or you might try:

```
MAKEDIR.BAT:
  CD %1
  ECHO CD %0>%1.BAT
  ECHO IF EXIST  AUTOEXEC.BAT
        AUTOEXEC>>%1.BAT
```

This command creates a subdirectory for you plus a Batch file of the same name that contains the MYSTERY.BAT statements described earlier. Try *MAKEDIR JUNK*, then type *JUNK*.

Notice the use of >> in MAKEDIR.BAT. It causes PC-DOS to append rather than overwrite to the standard output file. In other words, to create a multiline file, use normal redirection (>) for the first *ECHO* operation and appending redirection (>>) for all subsequent ones.

Commanding procedures

PC-DOS has an inscrutable little command called—of all things—*COMMAND*. It invokes a nested shell. What use is this without multitasking? Most people haven't the foggiest idea. But it is your key to calling Batch procedures.

This is trickier than it might seem. You can't just call a Batch file by name, as in

```
WONTWORK.BAT:
  ECHO About to call.
  PROC I'm a procedure.
  ECHO I'm Back!
PROC.BAT:
  ECHO %1 %2 %3
```

What goes wrong here is that PC-DOS invokes PROC.BAT but never returns. Sorry, but that's the way PC-DOS works. One Batch file cannot call another; it can simply *GOTO* another. Look at MAKEDIR.BAT again. It uses this feature to transfer to the AUTOEXEC.BAT file if it exists. There's no need to return in this case because calling AUTOEXEC is the last thing that MAKEDIR does.

Fortunately, when you do need to return, the *COMMAND* command will save the day:

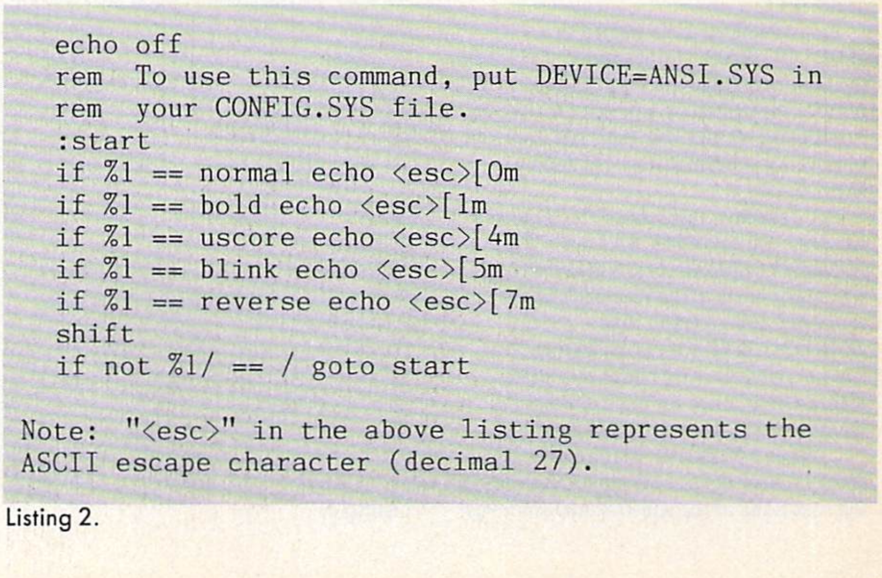
```
WILLWORK.BAT:
  ECHO About to call.
  COMMAND/C PROC
        I'm a procedure.
  ECHO I'm Back!
```

That's really all there is to it—just put *COMMAND/C* before the statement. Let's see what we can do with this:

```
M.BAT:
      ECHO OFF
      FOR %%F IN (%2) DO
           COMMAND/C X
             %1 %%F %3
X.BAT:
      ECHO OFF
      REM This is a procedure
           called by M.BAT
      CONFIRM %1 %2 %3?
      IF NOT ERRORLEVEL 1
           %1 %2 %3
```

To use this pair of files, you'll need the CONFIRM.COM program, which you can download from the BBS. The program displays its command line arguments as a prompt, waits for a Y/N answer, and returns *ERRORLEVEL* 0 for Y and 1 for N. This is one way to do keyboard input from a Batch file--though not the most powerful way, as we shall see.

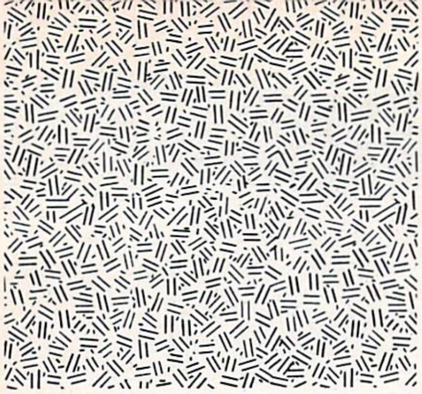
So what does M.BAT do? Well, "M" stands for "Multi" and you use it like this:



```
echo off
rem  To use this command, put DEVICE=ANSI.SYS in
rem  your CONFIG.SYS file.
:start
if %1 == normal echo <esc>[0m
if %1 == bold echo <esc>[1m
if %1 == uscore echo <esc>[4m
if %1 == blink echo <esc>[5m
if %1 == reverse echo <esc>[7m
shift
if not %1/ == / goto start
```

Note: "<esc>" in the above listing represents the ASCII escape character (decimal 27).

Listing 2.



```
M COPY A:*.* B:   or
M ERASE *.BAT   or
M TYPE *.TXT
```

For example:

```
A>M COPY A:*.* B:
COPY A:COMMAND.COM B:?n
COPY A:DEBUG.COM B:?y
1 File(s) Copied
COPY A:LINK.EXE B:?y
1 File(s) Copied
```

Recursive Batch

If you're ready for one of Batch's biggest surprises, check out a disk file called MAKEREM.LTG I've placed on the *COMPUTER LANGUAGE* BBS (415-957-9370), which was too long to be printed here. This quartet of Batch files expands on the idea behind *MAKEDIR*, giving you both a *MAKE* subdirectory command, MKS.BAT, and a remove subdirectory command, RMS.BAT.

The surprise is that RMS.BAT will remove not just the named subdirectory but also the entire sub-tree below that subdirectory, using recursion. This is one supercharged way to clean up your disk.

The secret of the recursive algorithm is on line 5 of MKS.BAT. Each time you invoke *MKS*, it makes a record of the subdirectory it creates for you by appending a line to a file called #DIRLOG.BAT. An instance of this file will be created in any subdirectory from which you execute MKS.BAT.

Just what does *MKS* record in #DIRLOG.BAT? Quite simply, it records the command that will later be needed to remove the directory you just created. By running the #DIRLOG.BAT file, you can remove all subdirectories created by MKS.BAT.

Let's see how this works. To remove a sub-tree you call RMS.BAT, as in *RMS JUNKDIR*. *RMS* first calls #RMS#.BAT, which connects to the root node of the requested sub-tree and runs #DIRLOG.BAT if it exists. In turn, this file removes all child nodes by calling #RMS#.BAT once for each child node. Here is where the recursion happens.

When #DIRLOG.BAT returns, #RMS#.BAT deletes all remaining files in the root node of the sub-tree and removes that node. Control then returns to RMS.BAT. Having removed a sub-tree, *RMS* must now update the current #DIRLOG.BAT file to delete the entry for that sub-tree. It uses the PC-DOS *FIND* filter to do this.

Study this use of *FIND* carefully. It lets a Batch file delete a desired line from a file. This is a good example of how combining Batch files with filters amplifies the power of both.

Environment control

Having made it this far, you are becoming a true Batch initiate. You are ready for one of Batch's greatest secrets— The Undocumented Feature. But first some background.

PC-DOS has a command called *SET* that lets you create string variables and assign them values. Saying *SET X=AHA* creates a variable *X* whose value is *AHA*. But PC-DOS makes almost no use of this capability. Is the environment just a gimmick?

Not at all. What IBM didn't tell you is that Batch files can refer to environment variables much like command line arguments. To refer to a variable *X*, simply put *%X%* in your Batch file. This expression will be replaced by the string value of *X* if *X* has a value, otherwise it will be replaced by the two-character string *X%*.

Right off the bat we can do something useful with this. Put the statement *PATH %1;%PATH%* in a file called ADDPATH.BAT. Typing *ADDPATH C:¼WORKDIR* will append that path name to the current search path. This works because PC-DOS records the current search path in an environment variable called *PATH*.

We can also use the string variables for true intrafile subroutines:

```
SET RET=L1
GOTO SUB
:L1
SET RET=L2
GOTO SUB
:L2
(etc.)
:SUB
ECHO I'm a subroutine
GOTO %RET%
```

```
echo off
rem This file demonstrates the intrafile subroutine checksub.
rem To call it, do:
rem SET ARG = value to check
rem SET VOC = vocabulary list to check against
rem SET RET = label to return to if check is OK
rem SET ERR = label to return to if check fails
rem Example:
set VOC=one two three 1 2 3
set ARG=%1
set ERR=EXIT
set RET=L1
goto CHECKSUB
:L1
Echo %1 is valid.
goto exit
rem Here is the subroutine
:CHECKSUB
for %%a in (%VOC%) do if %%a == %ARG% goto %RET%
echo "%ARG%" is invalid, try one of "%VOC%"
goto %ERR%
:EXIT
for %%v in (ARG VOC RET ERR) do set %%v=
```

Listing 3.

Listing 3 shows a more useful subroutine that validates a passed argument against a passed vocabulary.

Talking back to Batch

Now for the finishing stroke. String variables are the gateway to true interactive input to Batch files. The program SETVAR.COM, available on the BBS, makes this possible. *SETVAR* reads a line from standard input and assigns this as the value to the string variable named on the *SETVAR* command line. Basically, it lets you talk back to your Batch files.

For example, the following will prompt you for a file name and assign that name to a variable *F*:

```
ECHO Enter a file name.
SETVAR F
```

And the following will let your Batch file prompt for a missing command line argument

```
SET ARG=%1
IF NOT %ARG% = = ARG%%
    GOTO OK
ECHO (Your prompt for the missing
    argument)
SETVAR ARG
:OK
```

The next line will set a variable *CURDIR* equal to the path name of the current directory:

```
CHDIR |SETVAR CURDIR
```

Notice how we use the PC-DOS piping capability to feed the output from *CHDIR* to *SETVAR*. With pipes (|) you can take a program's output and assign it to a string variable, provided your environment has the room.

What if you'd like to check in advance whether there's sufficient room? Then simply *SET* your your variable to a test value and check whether that value was *SET* successfully:

```
SET MYVAR=0123456789
IF NOT %MYVAR% = = 0123456789
    GOTO NOROOM
ECHO There's room for at least a
    10-char value
```

For still more error checking, you can test the *ERRORLEVEL* after calling *SETVAR*. It will contain the length of the value string assigned by *SETVAR*, or it will contain 0 if the value string was null or couldn't be assigned.

The complete Batch

By now I hope you're hankering to try your hand at Batch programming. Maybe you've even got some ideas for programs like *CONFIRM* and *SETVAR* that you'll write yourself.

This is perhaps the nicest feature of Batch. Every program you add to your system becomes a verb of the Batch language. More than most others, you can truly make Batch your language.

To give you that extra dash of inspiration, I have another trio of Batch files to offer that constitute a menu generator (see the file MENUGEN.LTG on the BBS). Type *MAKEMENU* and you will be asked a set of questions that will lead to the creation of a brand new Batch file that implements a personalized menu.

You will specify the name of this menu file, the caption for each menu item (there are always nine), and the sequence of commands to be run for each item. You can specify any PC-DOS or Batch commands that don't use the equal sign (another quirk of PC-DOS). When you finish, you'll have an interactive menu of your own design to play with.

Batch files that write other Batch files? Is this artificial intelligence? An expert system? A fifth generation language? Hardly. It's just a little surprising. ■

Darryl Rubin is the network products section manager at Rolm.

Is your software easy to use?

Maybe YOU think so.

Despite what you may think, there may be times when your software is hard to use. **But what if your software could tell users exactly what to do every time they were confused?** Then people would start to agree with you about "easy to use".

SoftDoc™ is a module which provides your software with instantaneous context-sensitive help, on-line reference facilities and interactive tutorials. SoftDoc™ is compatible with windowing, networking, touch screens, mouse control and other interface technologies.

And when everyone agrees with you, prospective customers become eager buyers, first-time users turn into confident users, reviewers give you high marks, and sales people enjoy demonstrating your software. **Quite simply, when everyone agrees your software is easy to use, it sells.**

Call or write Learning Tools today. Find out how your software can become easier to use, demonstrate, learn, document, maintain, distribute, network, support, review and sell. And ask for a SoftDoc™ demonstration disk.

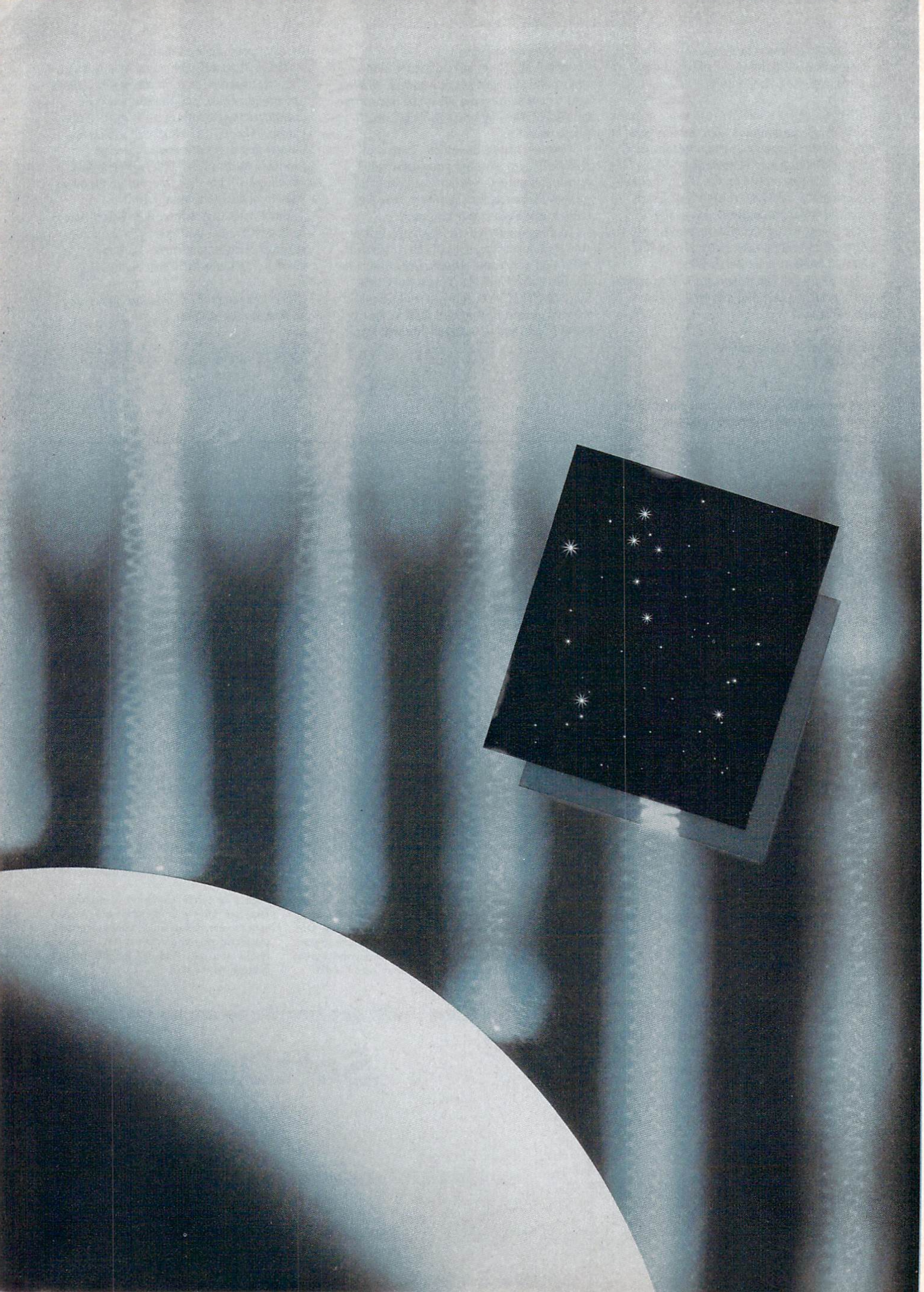
SoftDoc™ will get people to agree with you.

SoftDoc™

by LEARNING TOOLS

LEARNING TOOLS 686 Massachusetts Ave. Cambridge, MA 02139 Tel. (617) 864-8086

CIRCLE 37 ON READER SERVICE CARD



The Evolution of ZCPR

ZCPR's founder reveals the technical design behind this CP/M CCP replacement

By Richard Conn

At first a ZCPR3 System appears to operate like CP/M 2.2. The CP/M commands seem to work correctly—a *DIR* command will display the directory and a *TYPE* command will print a file out on the console. WordStar, dBASE II, BDS C, PASCAL/MT+, and all CP/M-compatible programs can run on it.

You may have heard about ZCPR3 on a local bulletin board system, in a chat with some friends at the last computer club meeting, or in a magazine article. Maybe ZCPR3 sounded like something special, something that could perform a variety of functions vanilla CP/M could not. And since it's been virtually free, you may have thought, "Why not try it—what's there to lose?"

Perhaps some time—after all, there are fourteen 8-in. disks of software to wade through, a 150-page installation manual, and a 500-page book (which should be available in October 1984), so it would take a little time to figure out what you have.

Is all this worth it? I think the answer is yes, but I'm prejudiced—I wrote ZCPR3 and use it all the time, including in the writing of this article.

The ZCPR3 System (Z80 Command Processor Replacement, Version 3) is a collection of tools based around the ZCPR3 command processor, which is a program that replaces the console command processor of CP/M.

ZCPR3 is an environment both upwardly compatible with CP/M 2.2 and extensible and adaptable to a variety of uses. It is an environment that can be used for software development and applications, and its tools can serve to increase programmer and user productivity. Its toolset, containing over 100 commands, provides a number of conveniences not available under CP/M and raises the user to a higher level of abstraction, further from the details of the machine and able to concentrate on the problem at hand.

The ZCPR3 System has been a source of excitement to many CP/M owners, but some of them still view ZCPR3 from the point of view of CP/M 2.2 or ZCPR2, its predecessor.

In some cases the philosophy of ZCPR3 is being missed, which is understandable with the current lack of documentation. In order to obtain the greatest benefit from using a ZCPR3 system, the philosophy of ZCPR3 should be understood.

Part I of this article will compare CP/M and ZCPR3's memory maps and cover the major ZCPR3 concepts. Part II, which will be in the November issue of *COMPUTER LANGUAGE*, will carry an in-depth technical discussion of the ZCPR3 front-end processor philosophy as a sample of some of the detail in ZCPR3 concepts, a brief overview of the tools available in the ZCPR3 distribution, and a section on where to look for more information.

CP/M and ZCPR3 memory maps

Figure 1 shows the memory maps of a conventional CP/M 2.2 system and a ZCPR3 system. From the point of view of a program designed to run under CP/M 2.2, both CP/M 2.2 and ZCPR3 look the same. From the point of view of a program designed to run under ZCPR3, ZCPR3 offers many more capabilities to this program than CP/M 2.2, and some programs intended to run under ZCPR3 simply cannot run under CP/M 2.2. ZCPR3 is upwardly compatible with CP/M 2.2.

Major ZCPR3 concepts

The sacrifice in scratch area for programs and data made by using ZCPR3 is rewarded by an increase in functionality and utility over CP/M 2.2. Some of the features added to the ZCPR3 System include:

- Extensions to the CP/M directory concept
- Extensions to the CP/M command processing algorithm
- Multiple commands on a single line and chaining
- A command search hierarchy
- A command search and directory search path
- Command scripts
- An integrated command file monitor
- An Environment Descriptor.

Directories. ZCPR3 allows the user to reference directories by disk letter, user area, disk and user area, or by name in the same way CP/M allows the user to reference disks. Two forms of directory reference—the DU (Disk/User) and DIR (named DIRectory) forms—are permitted.

The DU form is simply a specification like A: (for disk A, current user), 15: (for current disk, user 15), and A15:. The DIR form is a name that has been assigned a disk/user area, like JEFF: being assigned to B5:. The DIR form incorporates password protection as well as a directory reference, and when a DIR form is used as a prefix to a command or prefix to one of the two file name tokens, a check is made to see if an associated password has been defined, and the user is forced to provide this password before the reference is resolved.

Figure 2 illustrates some ZCPR3 commands using the available directory reference forms. The prompt is the full ZCPR3 prompt, which indicates disk, user area, and name of the current directory.

Commands. In a ZCPR3 system, commands can be found in four places:

- Within the ZCPR3 command processor itself
- Within memory-based resident command packages
- Within memory-based flow command packages
- In the form of .COM files on disk.

Like the CP/M 2.2 CCP, the ZCPR3 CP (Command Processor) contains some resident commands. It can contain all of the CCP commands (except *USER*, which is not needed anymore), but all of the ZCPR3 resident commands are different in one way or another from their CP/M counterparts. For example, the *TYPE* command stops after filling a page and allows the user to strike any key to continue, and the *ERA* command has an inspection option.

Commands may also be found within resident command packages. An RCP is a file (containing one or more commands) that is loaded into memory by the LDR tool of ZCPR3. Commands in an RCP are executed directly by the ZCPR3 CP without the need for disk references to locate and load the command. Each command in an RCP looks and acts like a .COM file, and a header exists at the front of each RCP which tells the ZCPR3 CP the names

```
                      CP/M 2.2 and ZCPR3 Memory Maps

        Address   CP/M System               ZCPR3 System

        High  --> ---------------------     -------------------------------
        Memory   | BIOS              |     | Various Packages and Buffers |
                  ---------------------     -------------------------------
                 | CP/M 2.2 BDOS     |     | Modified BIOS                |
                  ---------------------     -------------------------------
                 | CP/M 2.2 CCP      |     | CP/M 2.2 BDOS                |
                  ---------------------     -------------------------------
                 | Scratch Area      |     | ZCPR3 Command Processor (CP) |
                 |                   |      -------------------------------
                 |   for Programs    |     | Scratch Area for Programs    |
                 |                   |     |   and their Data             |
                 |   and their Data  |     |   (1K-5K smaller than under  |
                 |                   |     |     CP/M 2.2)                |
           100H -> ---------------------    -------------------------------
                 | CP/M Buffers      |     | ZCPR3 Buffers                |
             0H -> ---------------------    -------------------------------
```

Figure 1.

```
                      Some Directory Reference Examples
     B7:TEXT>5:                    log into user 5 on disk B
     B5:BASIC>TEXT:                log into the directory named TEXT
     B7:TEXT>DIR ROOT:             obtain a directory display of ROOT:
     Password? MYSYS               ROOT has a password on it
          ( display follows )
     B7:TEXT>COMS:DEMO             run DEMO.COM from directory COMS:
          ( program runs )
     B7:TEXT>PRINT B5:*.*,SCR:*.*  print all files in B5: and SCR:
          ( printout occurs )
```

Figure 2.

of the commands in the RCP and their locations. RCPs offer several advantages:

- Disk space can be saved because a number of small commands can be grouped together in one file and loaded for execution as a group.
- Time is saved because RCP-loaded commands are memory-resident once their RCP has been loaded, so no disk activity is involved in locating and loading them.
- Commands normally found in the ZCPR3 CP, like *DIR* or *TYPE*, can be placed into an RCP, freeing up the CP for more system-oriented functions and giving more room to add more features to the resident commands.
- Commands residing within an RCP do not affect the transient program area as a rule, so debugging facilities can be placed into RCPs to look at the TPA after a program has executed there.

Several standard RCPs are included in the ZCPR3 distribution files, and some of the commands they provide include:

- *CP*—Copy a file. Example: *CP ROOT:NEWFILE.TXT=B5:T.TXT*
- *MU*—Memory utility, a screen-oriented memory editor that allows the user to display and change memory without affecting the TPA by its invocation
- *P*—Dump memory (Peek) without affecting the TPA. Example: *P 100 2FF*
- *POKE*—Change bytes in memory. Example: *POKE 10F A 1 2* to change the bytes at 10FH to 111H
- *PROT*—Set file protection. Example: *PROT *.* R* for making all files in the current directory read-only
- *TYPE*—Improved *TYPE*. Example: *TYPE*.TXT*

A third source of commands is a flow command package. An FCP is very similar to an RCP in that it is a package of commands loaded by LDR and executed directly from memory by the CP.

Commands that control the flow state of the ZCPR3 System (like *IF* and *ELSE*) are located here since these commands will be executed regardless of the status of the flow state (see the command search hierarchy). The ZCPR3 CP is aware of the flow state of the system; if this state is TRUE, the CP will allow any accessed command to execute. If the flow state is FALSE, only commands resident within an FCP may be executed, and all other commands are flushed without error.

```
                    The File ASM.SUB

     ASM $1.BBZ                        assemble
     IF INPUT
     LOAD $1                           load and cleanup
     ERA $1.HEX
     ELSE
     ERA $1.HEX                        else just cleanup
     FI                                end of IF
     ERA $1.BAK                        cleanup more
```

Figure 3.

```
           Running the Command "SUBMIT ASM MYFILE"

ASM MYFILE.BBZ                  assemble a program

IF INPUT                        raise to the next flow state
                                 and allow the user to set
                                 it to TRUE or FALSE
LOAD MYFILE                     run LOAD MYFILE and erase
ERA MYFILE.HEX                   MYFILE.HEX if TRUE

ELSE                            ... otherwise ...

ERA MYFILE.HEX                  just erase MYFILE.HEX

FI                              done with IF

ERA MYFILE.BAK                  erase backup file
```

Figure 4.

```
          MCL Examples (Spaces Added for Clarity)

B4:PASCAL>      PASCAL MYFILE;  LINK MYFILE
B7:TEXT>        FORMAT MYFILE.TXT;  PRINT MYFILE.FRM
B0:SCR>         B:;  ERA *.BAK;  DIR;  C7:;  PRINT *.TXT
```

Figure 5.

Nine flow states may exist at any one time in a ZCPR3 System—the empty state (which is TRUE) and the *IF* Levels 1 to 8. The *IF* command is used to raise the system to the next flow state and set this state to TRUE or FALSE. The *FI* (*End IF*) command is used to drop down to the previous flow state. The *ELSE* command toggles the value (TRUE/FALSE) of the current flow state. And the *XIF* (*Exit all IFs*) command terminates all *IF* levels (forces the TRUE empty state) if the current *IF* level is TRUE and does nothing otherwise. Since there are eight *IF* states, IFs may be nested up to eight levels deep (Figures 3 and 4).

The fourth command source is the standard .COM file, for example, WordStar (WS.COM) or dBASE II (DBASE.COM). ZCPR3 handles the execution of .COM files like CP/M does, but ZCPR3 locates .COM files by searching for them unless the user explicitly tells it not to.

Multiple command lines. The MCL is a third major feature of ZCPR3. ZCPR3 allows the user to specify a sequence of commands, separated by semicolons, to be executed on one line (Figure 5). This feature buys the ZCPR3 user two major advantages:

- A sequence of commands can be issued at one time, and the user can go off and do something else while they execute.
- One program can invoke another by placing a command line into the MCL buffer, setting a pointer to the first character of this line and returning to the operating system.

Command search hierarchy. Each time a command is processed, the ZCPR3 CP follows a sequence of steps in searching for the source from which the command will run. This is the command search hierarchy, and the hierarchy is:

1. Input and parse the next command in the MCL buffer
2. Check the current FCP for the command and run it if the FCP contains the command
3. Check the flow state; if TRUE, continue, or, if FALSE, flush the command and advance to the next one (return to step 1)
4. Check the current RCP for the command and run it if the RCP contains the command
5. Check the ZCPR3 CP for the command and run it if the CP contains the command
6. Search along the command search path for a .COM file, logging into directories until either the file is found or the bottom of the path is reached; load and run the .COM file if found
7. If an extended command processor has been specified (at installation time), load it and pass the command to it for execution
8. If steps 4-7 fail, invoke an error handler if one has been installed; if none installed, print the COMMAND? error message.

The ZCPR3 CP follows these steps each time a command in the command line buffer is resolved. Thanks to the fact that steps 1-5 involve the use of memory-resident facilities and step 6 incorporates an efficient directory search algorithm, the procedure of following the command search hierarchy takes very little time.

For the sake of space, extended command processors and error handlers will not be discussed in this article. The reader is referred to the book *ZCPR3: The Manual* to learn more about these topics (see section on where to look for more information in Part II of this article, appearing next month).

Command search path. A path under ZCPR3 is an expression of a sequence of directories, and a command search path is the directory sequence to be followed when the ZCPR3 CP is looking for a .COM file. Paths are implemented as a sequence of DU forms, where the dollar sign ($) is used to extend the DU form to indicate the current disk or user area. For example,

```
$$ $0 A$ A15
```

indicates the path from (1) the current disk and user area to (2) user area 0 on the current disk to (3) the current user area on disk A to (4) disk A and user area 15. If the user is logged into B7, this path is translated into

```
B7 -> B0 -> A7 -> A15
```

The *PATH* command under ZCPR3 is used to define the command search path and change it while the user is running the system (Figure 6).

Scripts. In many situations, the user ends up issuing the same sequence of commands, perhaps with minor variations, over and over again. For instance, the user may want to assemble a program, check for errors, and, if no errors, link it and create a .COM file. If ASM were the assembler and ASM were the file, then the command sequence might be:

```
ASM $1.BBZ
< if no errors > LOAD
ERA $1.HEX
```

The ZCPR3 System provides a convenience to assist the user in cases like this—the script or *alias*. An ALIAS is a .COM file created by the *alias* tool which contains a command sequence that is executed when the *alias* is called. Continuing the above example, the following ALIAS could be created:

Sample PATH Commands (Current Dir is B12)

Path Command	Description	Example
PATH A$ ROOT	set path from current user on disk A to the directory ROOT	A12 -> ROOT
PATH A$ $2 C7 ROOT	set path from current user on disk A to user 2 to current disk to disk C, user 7 to ROOT	A12 -> B2 -> C7 -> ROOT

Figure 6.

```
ASM80:
     ASM $1.BBZ;   assemble program
     IF INPUT;     allow user to ap-
                   prove continue
     LOAD $1;      convert HEX to
                   COM
     FI;           end of IF
     ERA $1.HEX
```

By issuing the command *ASM80 MYFILE*, this sequence of commands is run:

```
1.  ASM MYFILE.BBZ
2.  IF INPUT
3.  LOAD MYFILE
4.  FI
5.  ERA MYFILE.HEX
```

An *alias* can be employed in four ways under ZCPR3:

- An *alias* is usually run on cold boot to execute a series of programs that initialize the system.
- The *CD* command, which is another way to log into a directory, will automatically run the *alias* ST.COM if one is found in the directory being logged into. For instance, *CD ROOT:* will log the user into *ROOT* and run the command ST.COM if one is located in the directory called *ROOT*, thereby establishing an entirely new operating environment. Issuing the command *CD ROOT:* is the same as issuing a sequence such as,

```
ROOT:;
< if ST.COM exists in ROOT > ST
```

- Commonly-used command sequences, like the *ASM80* example above, can be stored in the directory at the end of the command search path for execution from any directory on the system. The last path element, which is recommended to be the *ROOT* directory, should contain an absolute directory reference, like A15 or A0.
- Command sequences used over and over for a particular need at a particular time may be quickly placed into an ALIAS and the *alias* command may be run over and over rather than repeatedly typing the command sequence. For example, if the user is making several copies of a set of files on different disks and printing a directory listing for each disk, an ALIAS containing the command sequence

```
MCOPY BACKUP:=*.*;
     XDIR BACKUP: P
```

would copy the files to the directory named *BACKUP* and then print a directory display on the printer of this disk. If this sequence was set up as an ALIAS named *CPY*, then five disks could be copied by issuing the command *CPY* five times rather than issuing the above sequence five times. The same kind of thing could be done with SUBMIT files, but an ALIAS has certain capabilities that SUBMIT does not (see the references).

ZEX command file processor. ZEX, which stands for Z80 Executive, is an integral part of the ZCPR3 System, and it provides a memory-based command file facility similar to SUBMIT but stores the commands in a memory buffer and executes them directly from the buffer via the memory-resident ZEX monitor.

Unlike SUBMIT, ZEX is integrated into the system, and a program (like a .COM file loaded into the TPA) can communicate with ZEX directly, looking at the commands it is about to issue and changing the command flow within the ZEX command file (implementing a *GOTO* statement).

ZEX provides information about its state to the ZCPR3 System through the Environment Descriptor, and a program can read this information and find out and change where the next character ZEX is going to input comes from, where the first character in the command file is, and how to turn the ZEX monitor on and off to control ZEX operation.

Environment Descriptor. Under CP/M 2.2, a few simple features of the design (like the BIOS and BDOS entry points and the FCB and 80H parser buffers) allow the very useful capability of transportability of binary files between different CP/M systems to be possible. With this capability, the door for the development of the CP/M world was opened. Software engineers and programmers could create software that would run on any CP/M system regardless of the hardware configuration. A market was created by this virtual machine of CP/M.

In remaining compatible with CP/M 2.2, these simple features were retained in the design of ZCPR3 in their entirety with

NGS FORTH

A FAST FORTH OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MSDOS COMPATIBLES.

*79 STANDARD

*FIG LOOKALIKE MODE

*PC-DOS COMPATIBLE

*ON-LINE CONFIGURABLE

*ENVIRONMENT SAVE & LOAD

*MULTI-SEGMENTED

*EXTENDED ADDRESSING

*AUTO LOAD SCREEN BOOT

*LINE AND SCREEN EDITORS

*DECOMPILER & DEBUGGING AIDS

*8088 ASSEMBLER

*BASIC GRAPHICS & SOUND

*NGS ENHANCEMENTS

*DETAILED MANUAL

*INEXPENSIVE UPGRADES

*NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICE: $70

PLEASE INCLUDE $2 POSTAGE & HANDLING WITH EACH ORDER.

CALIFORNIA RESIDENTS : INCLUDE 6.5% SALES TAX.

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

CIRCLE 45 ON READER SERVICE CARD

few changes. ZCPR3, however, offers the Environment Descriptor as an additional feature that opens many doors to the software developers and extends the virtual machine of CP/M. With the Environment Descriptor, the following additional information is made readily available to any program running under ZCPR3:

- Attributes of the user's CRT terminal, as in the width of the screen in characters and number of lines on it
- Attributes of the user's printer, including a flag indicating if it can form feed, the number of lines on the printer, and the maximum width of the lines
- The locations of the FCPs, RCPs, message buffers, ZEX control buffers, named directory buffers, and other ZCPR3-specific buffers
- A terminal capabilities data record that describes the sequences used to clear the CRT screen, position the cursor, enter highlight mode, etc.

Under ZCPR3, transportable programs such as screen-oriented editors are possible, and these programs can be moved from one ZCPR3-based computer to another as binary images, with the only installation requirement being for the placement of the address of the Environment Descriptor at a standard location within the program.

The utility Z3INS is provided to perform this installation, which it does very fast. Installation of the 58 .COM files in the first phase of the ZCPR3 release takes about three minutes via Z3INS. The full ZCPR3 distribution includes several screen-oriented tools that dramatically illustrate the power of this extension to the CP/M concept.

Next month, I'll elaborate further on the features of ZCPR3. For more immediate information and access to ZCPR3 programs and documentation, call the *COMPUTER LANGUAGE BBS* (415-957-9370) and leave me a message. See you next month! ■

Richard Conn has a B.S. and M.S. in computer science. His current interests include operating systems, C and UNIX, and the Ada programming language.

TOTAL CONTROL:

***FORTH:* FOR Z-80®, 8086, 68000, and IBM® PC**

Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS
DATA ACQUISITION • PROCESS CONTROL

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth + and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities and 200 page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, $100.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $100.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH +	$250.00
8086 FORTH + for CP/M-86 or MS-DOS	$250.00
68000 FORTH + for CP/M-68K	$400.00

Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

VISA

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to (213) 306-7412

MasterCard

CIRCLE 35 ON READER SERVICE CARD

PROFESSIONAL BASIC™

Access All The Memory Of Your PC!

Professional BASIC™ - the first of it's kind: a powerful programming language for the 16-bit microcomputer. You can simultaneously view program execution and see the change in variables, the code being executed, file buffers, or an "instant replay" of execution within the multi-window environment.

"The real magic of Professional BASIC™ is its wealth of 'windows' into an executing program." "I'm frankly amazed; My hat is off to Dr. Bennett...An elegant piece of coding indeed."
Personal Computer Age Dan Rollins, Feb. 1984

"Professional BASIC™...cranked out a timing and absolute error result competitive with the best compilers, while simply blowing the doors off every other known microcomputer interpreter."
Dr. Dobb's Journal Ray Duncan, Aug. 1984

FEATURES

- Interpreter language.
- Runs most IBM® PC BASIC programs.
- 8087 numeric coprocessor support.
- Have a 200x200 array (160k).
- Semi-compiles for immediate error checking and faster execution.
- Character-by-character syntax checking.
- More than 18 trace (half/full) windows.
- BCD arithmetic option, with 8087 (16 digit accuracy).
- Labeled lines. Use of 'GOTO label' statements.
- Save programs (or subroutines) without line numbers.
- Cross referencing of variables.
- 'SEARCH text' command.
- Large real numbers (10^{308} to 10^{-308}).
- Large integers (over 2 billion).
- Large files (over 4 billion records).
- Complete-sentence error messages.
- Runs on the IBM® PC and other compatibles.

PROFESSIONAL BASIC™
only $345

See Your Local Dealer
Demo Disk $5

Morgan Computing Co.
10400 N. Central Expwy., Suite 210
Dallas, Texas 75231
(214) 739-5895

CIRCLE 42 ON READER SERVICE CARD

The four commandments

MNSNUS (or, Using Mnemonic Atoms in Symbolic Naming)

By Ron Gutman

Assembly languages and high-level programming languages free us from using numbers to tell our computing machines what to do. These languages also require us to devise many names—for variables and subroutines, for instance—ourselves. We typically choose mnemonic names derived from our human languages so that the names remind us of the activities they stand for.

Programmers appreciate this freedom to devise names to their own liking. But the freedom can be a burdensome responsibility, especially when writing a large program that contains hundreds or even thousands of user-defined names or symbols. How does a programmer select symbolic names that describe the objects to which the names are given?

Before inventing the technique this article describes, I struggled to choose names in a consistent fashion so that I could subsequently recognize the names and the objects they referred to. This took a lot of programming time, and the results were not the best. Often I would end up with the same name for two different things. Other times I could not recognize the name or mistook it for something else. Could I expect anybody else to understand the names I chose?

Contributing to the problem, the assembler I used allowed only six characters to form each symbolic name. A limitation that severe is not uncommon in assembly languages and comparably severe limitations exist in some compilers for high-level languages.

How does a programmer describe an object in six characters? I often felt that I needed six words. It doesn't help much that the assembler or compiler allows you to tag on extra characters beyond those it uses to distinguish one symbolic name from another. You still have to insure that each name you choose is distinguished from all others in the program you are writing.

My story does have a happy ending, though. I overcame the problem by inventing a technique using "mnemonic atoms," as I call them.

If readers find themselves in a situation similar to mine—having to write a large program containing a lot of somewhat abbreviated symbolic names—the technique I will describe may be invaluable to their projects. This is particularly the case if readers use assembly language because its programs can have large numbers of symbol definitions. There are fewer facilities (like *GOTO*-less control structures) that help avoid the use of symbols. My technique would also be useful where storage limitations demand a terse source code. Since mnemonic atoms provide a degree of documentation, they can reduce the need for comments in the code.

Of course, one solution to the problem is to use programming systems that are less restrictive with regard to user-defined symbols. COBOL programmers can use—and usually do use—long-winded names like "monthly-income-average." A COBOL programmer can select a few words of English or computer jargon that describe the thing he or she is naming and put them together with hyphens.

When a programmer is limited to six characters, he or she will do the same thing but will abbreviate each word and omit the hyphens. So instead of "monthly-income-average" the programmer might use *MOINCA*. Or, perhaps unsatisfied with such short abbreviations, the programmer might choose to leave out one of the components, arriving at *MOINCO* for "monthly-income". *MO* seems like an adequate abbreviation for "month" because it is generally accepted as such, but *INCO* is more recognizable as "income" than *INC*, which could be confused for other words such as "increment".

Now let's pretend we are browsing through a source listing and see how we do at recognizing some abbreviations. We see a reference to a subroutine called *SRTBL*. That could be an abbreviation for "search-table". Or is it "sort-block"? Of course, the context or the comments will tell us, but recognition does not come quickly.

Suppose *SRTBL* searches a table that contains names and phone numbers for a given name, and suppose we find that we also need the capability to search the table for a given phone number. So we write a new routine that we will call *SRTBLP* to search by phone number (*P* for "phone"). The existing routine we rename *SRTBLN* to make it clear that it searches by name (*N* for "name").

Later we find that we want to sort the table and print it out either in alphabetical order by name or numerical order by phone. If *SRT* is our abbreviation for "sort", then we ought to call our new routines *SRTTBLN* and *SRTTBLP*. But now our assembler, which only recognizes six characters of each name, considers these two names to be identical or just plain illegal. We can't use *SR* for "sort" because we have already used the names *SRTBLN* and *SRTBLP* for our search routines. We finally settle on *SRTTBN* and *SRTTBP*, using *TB* for "table". Having two abbreviations for "table" is a little disconcerting, however. Maybe we should rename our other routines to use the new abbreviation for "table".

Some fiddling around would yield more solutions—each requiring some kind of compromise—but the reader may now have a feel for some of the problems involved in choosing names. After our efforts at solving this naming puzzle, how well will our solution serve us? Will we be able to recognize *SRTBLN* and *SRTTBN* and be able to say readily which is which? How about a month from now, when we might have to modify the program?

It would be difficult, primarily because the abbreviations are sometimes one, sometimes two, and sometimes three characters long. How will our eyes divide these names into their components? By trial and error. Remember that *SRTBLN* is divided *SR-TBL-N*, while *SRTTBN* is divided *SRT-TB-N*, or did you forget already? Seeing the *SRT* in *SRTBLN* will certainly distract us from its correct division.

Don't think these issues are too trivial. They matter in all but the smallest programming projects.

So how can we put some order into this naming business where there is now confusion?

At this point I think our resistance to applying some kind of disciplined scheme to the problem has broken down, so I will now offer my rules for constructing mnemonic names. The first two are commandments to be followed religiously. They will make all the difference in the world. The other two will help, but they need not be followed religiously because sometimes a given situation just won't oblige us in our efforts to follow them.

The First Commandment

All abbreviations shall be composed of the same number of characters. These equal-length abbreviations are called mnemonic atoms.

All mnemonic atoms have the same number of characters. Fine, but what is that number? You have to make a trade-off between the number of possible mnemonic atoms you can invent and the number you can use to form one name. You want both of these numbers to be as large as possible. However, your programming language limits the number of characters in a name, so if your atoms contain too many characters your symbolic names won't contain very many atoms. On the other hand, if your atoms contain too few characters, there won't be many combinations of characters to form atoms.

Suppose names are limited to six characters, and suppose you decide that all of your atoms will be two characters long. Then you can combine up to three atoms to form a name and, assuming you use only alphabetic characters, there will be 26 x 26, or 676, sequences of two characters you can choose from to form atoms. I think two characters per atom is optimum given a limit of six characters per name.

If pressed for a formula, I would venture that the number of characters per atom should be the square root of the character limit rounded down to the nearest whole number. By that formula, two-character atoms would be used with any limit from four to eight characters, and three-character atoms would be used from nine to 15. Above 15, you might prefer to use another scheme altogether and form your names in the COBOL fashion using a delimiter to separate the components.

The Second Commandment

Each mnemonic atom shall be entered into a "dictionary" of mnemonic atoms. This dictionary shall be in alphabetical order and must give the meaning of each.

Perhaps the word "dictionary" implies too much tedium. But my dictionaries have been just a few pages long. I include my dictionary in the source code as a section of comments, or I maintain the dictionary in a separate text file that later becomes part of the documentation in the software package.

In any case, use the computer to maintain the dictionary. You won't invent all of your atoms at one time. You might start with a few atoms in the dictionary, but you will be adding them as you go along. One of the main purposes of the dictionary is to tell you those combinations of characters that you have already used before you attempt to add new ones. When I add an atom, I just pencil it into my dictionary listing with an arrow to show where it goes. Periodically I edit the dictionary on the computer to get a clean, up-to-date listing.

Your dictionary must be alphabetized by mnemonic atom. Otherwise it will be intolerably tedious for you to determine what new atoms you can add or find out what an unfamiliar atom means.

Now for an advanced lesson. Divide your dictionary into two dictionaries. One dictionary will contain atoms that you are likely to use in future projects. Most of these atoms will be abbreviations for common programming terms like "table", "stack", "pointer", "index", "move", or "error". The other dictionary will contain atoms that are specific to your current project. These atoms will be abbreviations for terms used in the specific application you are working on. When you go on to your next project, you can start out with the dictionary of common atoms, leaving your application-specific dictionary behind. The big pay off comes when you use atoms you have already become familiar with.

A brief example of a typical dictionary appears in Table 1.

The Third Commandment

Do not create two mnemonic atoms with identical meanings, such as SR *for "search" in some cases and* SE *for "search" in others. Conversely, avoid using one mnemonic atom for more than one one meaning, such as* SR *for "search" in some cases and "sort" in others.*

There are two purposes in not creating two atoms with the same meaning. One is to keep to a minimum the number of atoms your mind has to deal with—thereby increasing your facility with them. The other is to keep to a maximum the number of unused combinations of characters available for new atoms. Often the combination you want for a new atom has been used, making it hard to follow the second part of the commandment. This will happen less often if the first part is kept in mind.

It's not hard to see the benefit reaped when each mnemonic atom has only one meaning: less ambiguity in deciphering symbolic names in your source code. But it is hard to achieve this. Often the most mnemonically satisfying abbreviation of a term is already being used for another term (more on this matter later). Use the dictionary to check whether an abbreviation is already in use.

The Fourth Commandment

Be conservative about inventing new mnemonic atoms. Use an existing one if possible.

Put no frivolous atoms in the dictionary! This has the same purpose as The Third Commandment. A frivolous atom might be one you really can't use or one that serves virtually the same purpose as another atom. Also, avoid creating atoms for vague terms with broad meanings like "number", "process", or "data" unless you have a specific meaning for them in your application. You are trying to pack as much meaning into your atoms and the names built on those atoms as you can. In that respect, atoms for vague terms just don't carry enough punch.

An example

The second part of the dictionary in Table 1 contains atoms that might be used in a mailing list program. As the project progressed, this part of the dictionary would grow considerably as would the first part if the programmer were starting a dictionary from scratch.

Notice that some atoms are not defined by just one word. *DW* stands for "day of week". *PH* stands for "phone number". Do not think of atoms as having single-word definitions—though they often will—because that concept is too restricting.

Many atoms come in pairs that represent complementary concepts such as *MN* and *MX* for "minimum" and "maximum" or *RD* and *WR* for "read" and

Sample mnemonic atom dictionary

AK - acknowledge

CL - clear, reset
CP - copy
CR - create
CU - current
CV - convert

DA - disable
DB - debug
DC - decrement
DL - delete
DW - day of week
DT - date

EA - enable
EO - end of
ER - error
EX - exit

FI - file
FL - flag
FP - floating point value

IC - increment
IP - input
IR - interrupt
IS - insert
IV - interval, span of time
IX - index
IZ - initialize

LN - length
LS - list

ME - menu

MN - minimum
MO - mode
MS - message
MV - move
MX - maximum

OP - output

PK - pack
PN - pointer
PR - prompt
PV - previous

QU - queue

RC - receive
RD - read
RE - record

SE - set
SK - stack
SO - sort
SR - search
SV - save

TB - table
TD - time of day
TK - task

UP - unpack

WR - write
WT - wait

XM - transmit, send

The following atoms stand for terms and concepts related to the specific application of a mailing list program:

AD - address
NA - name

PH - phone number
PU - purge

ML - mailing label

Table 1.

C

Software Development

PCDOS/MSDOS

Complete C Compiler
- Full C per K&R
- Inline 8087 or Assembler Floating Point, Auto Select of 8087
- Full 1Mb Addressing for Code or Data
- Transcendental Functions
- ROMable Code
- Register Variables
- Supports Inline Assembler Code

MSDOS 1.1/2.0 Library Support
- All functions from K&R
- All DOS 2.0 Functions
- Auto Select of 1.1 or 2.0
- Program Chaining Using Exec
- Environment Available to Main

c-window™ Symbolic Debugger
- Source Code Display
- Variable Display & Alteration Using C Expressions
- Automatic Commands
- Multiple Breakpoints by Function & Line Number

8088/8086 Assembler
- FAST — Up to 4 times Faster than IBM Assembler
- Standard Intel Mnemonics
- Compatible with MSDOS Linker
- Supports Full Memory Model

8088 Software Development Package **$199⁰⁰**

Includes: C Compiler/Library, c-window, and Assembler, plus Source Code for c-systems Print Utility

c-systems

P.O. Box 3253
Fullerton, CA 92634
714-637-5362

"write" or *DA* and *EA* for "disable" and "enable". *EO*, for example, could be used in combination with *FI* or *RE* for "end-of-file" or "end-of-record".

Now we can apply mnemonic atoms to the hypothetical problem we tackled above. Those subroutines needed to search the table by name and by phone number (which we named *SRTBLN* and *SRTBLP*) can now be called *SRTBNA* and *SRTBPH*. The corresponding sort routines can be called *SOTBNA* and *SOTBPH*. This is a consistent and elegant scheme for naming these routines. We have given up *TBL* as an abbreviation for "table" for the consistent use of *TB*. It is no loss because as long as we must get used to *TB*, the use of *TBL* only adds confusion.

Of course, the atoms could be combined in a different order. I'm going to permit the issue of order to be decided by the reader. However, I will say that the ordering of atoms should be exploited to put more information into symbolic names, so the programmer should devise some ordering scheme.

Using my own personal scheme, the routine to search by name would be called *TBNASR*. That is because I like the last atom to reveal what kind of thing is being named. For variables and data structures, my final atom acts as a noun, as in *TBSRPN* for "table-search-pointer". For routines, my final atom is a verb. In the case of *TBNASR*, *SR* stands for the verb "search", which tells us that the thing being named is a routine that searches. The remaining atoms in the name are modifiers of the verb or noun. *TBSRPN* is a pointer. The atoms *TB* and *SR* modify the word "pointer" by telling how the pointer is used (it is used in the table search).

Similarly, if I had a name for the string variable that holds the name *TBNASR* is to search for, then the name would be *TBSRNA*. This last example shows how the order of atoms can be used to impart information. *TBSRNA* is just a permutation of atoms of *TBNASR*, but the order allows me to distinguish between the two.

This last example illustrates the ability of each atom to be used in different ways as a noun, verb, or modifier—an important feature of mnemonic atoms because it allows each atom in your dictionary to do more for you. This is possible because the atoms can be permuted in any fashion without affecting the way names are divided into atoms. That, in turn, is possible only because the atoms are equal in number of characters.

If the atoms were unequal—some one, some two, and some three characters—they could be permuted at the expense of having inconsistent division of names, or the names could be consistently divided into unequal atoms, but each atom could only serve in a particular position in any name. The advantages of consistent division and permutability are only achieved with equal atoms.

Often my first atom identifies a module with which the object being named is associated. I might have, for instance, a module that handles all of the operations on a table. Then *TBNASR*, *TBSRNA*, *TBPHSO*, etc. are all identified as being associated with that module by their initial atom, TB.

Again, the wise strategy is to devise a scheme that uses the order of atoms to convey information.

I'm going to leave one issue open because it is beyond the scope of this article and probably best left to personal preference. That is the nasty issue of selecting abbreviations to use for atoms. It is particularly hard to avoid conflicts when your atoms are only two characters. Some combinations of characters, such as *IN* and *ST*, just cry out to be used over and over. *IN* could be an abbreviation for "input", "index", "initialize", and many other terms. *ST* could stand for "status", "stack", or "store". Notice that the dictionary in Table 1 avoids these abbreviations altogether.

But the good news is that it doesn't matter much what abbreviations you come up with or how inappropriate they seem. If you use them consistently, they will become old familiar pals that couldn't seem more appropriate, just as "lbs." is a very familiar abbreviation for "pounds".

Use your imagination. The abbreviation doesn't have to be the first two letters of the word being abbreviated. Even the first letter does not have to be used. *NT* has mnemonic value as an abbreviation for "integer" because the sounds agree when we pronounce the abbreviation (in-tee) and pronounce the word "integer". And there is *XM* for "transmission" because "trans" means "cross". Also try to abbreviate a different word with the same meaning. *CL* can be used for "clear" instead of *RE* or *RS* for "reset".

One handy rule of thumb: when you have the choice, rely on the more uncommon letters (*J*'s, *Q*'s, *X*'s, and *Z*'s) in your abbreviations. They will present fewer future conflicts and will more readily bring to mind the terms for which they are abbreviations.

No doubt there is an article to be written on the art of abbreviation. If adopting mnemonic atoms causes the reader eventually to write such an article, I promise to read it.

Ron Gutman has a B.S. and M.S. in computer science from the Univ. of Calif. at Berkeley. He has been designing and implementing software for seven years and is now working for GRiD Systems Corp. in Mountain View, Calif.

CIRCLE 16 ON READER SERVICE CARD

Low-level Assembly Interface on the IBM PC

Speed up your BASIC and improve memory utilization with common assembly subroutines

By Jeri Girard

When high-level languages like BASIC, COBOL, and FORTRAN were invented, the average programmer was able to speed up his or her writing of code by a good 50%. The programmer, no longer needing to worry about inspecting bytes one by one or flipping bit switches, could concentrate on building program logic and interpreting problems instead.

The development of high-level languages in general, and BASIC in particular, was one of the major steps in making home computers acceptable to the programmer and hobbyist.

BASIC is easy to understand, simple to use, and relatively transportable between machines. But as everyone knows, "There's no such thing as a free lunch." There had to be a trade-off somewhere, and the trade-off was in speed. Interpretive BASIC is notoriously slow.

This speed problem doesn't matter much in most applications since a computer spends a lot of its time waiting for information. As Crayne's Law says, "All computers wait at the same speed," regardless of the language they're using.

The speed inefficiency of BASIC is most noticeable in game programs, where there is continuous action. The typical shoot-'em-down video games tend to run like slugs in BASIC. Text adventures, which do a lot of data base searching, need several seconds to respond to player commands.

Another problem with BASIC is its inability to use available machine memory. As implemented on the IBM PC, BASIC uses the first 64K above DOS. It is limited to that amount of memory and incapable of handling any BASIC program larger than 64K. Even if you've got a machine with 512K you're still stuck with BASIC's 64K work area (at the present time).

In order to get around time and space limitations, a lot of programmers are going back to machine language—at least for part of their code. By writing common subroutines in assembly language they are getting the speed and memory utilization of the low-level code while preserving the ease and familiarity of BASIC.

If you have worked with assembly language before, you've probably considered this approach yourself, but you may have gotten discouraged when you tried to develop the BASIC/assembly language interface. IBM's BASIC manual devotes an entire 18-page appendix to the subject, but the suggested procedures are cumbersome and require a lot of manual intervention by the programmer.

Although the systems shown in the manual will certainly work, one of them calls for converting each line of the subroutine into machine code, translating it into hex, and then using *POKE* to insert the instructions one by one into memory. This is a lengthy and time-consuming procedure. Another suggested way of loading a routine is to use *DEBUG* to load it into high memory, where it overlays the transient portion of COMMAND.COM. This requires you to reset the system registers and use the *DEBUG N* command to initialize the parameter passing area.

Besides the actual program load, another problem that has to be dealt with is deciding exactly where in memory to locate the new code. You have your choice of inserting it within the 64K BASIC work space or, if the BASIC program is too large to allow that, putting it somewhere else in memory. In either case you have to determine the end of BASIC itself, which can vary depending on the device drivers installed in DOS.

Using relative addressing

Fortunately there is a relatively easy procedure that will let you load assembly subroutines via interpretive BASIC without resorting to *POKE* and without having to use *DEBUG* to find the end of the interpreter work area. This procedure loads the code from within the BASIC program and allows you to invoke it from anywhere in the program with a simple *CALL* statement.

Let's take as an example a subroutine that converts an input text string to upper case. This procedure is done in countless programs to let the user enter either upper or lower case letters. Without it the user is either limited to all upper case input, or the BASIC program has to make two checks for each character entered. For the sake of this discussion let's assume you have written a program called SUBRTN.ASM, which includes the routine

XLAT (Listing 1). As you can see, this is a straightforward and simple routine.

It is invoked from BASIC with the command *CALL XLAT (A$)*, where *A$* is the name of the string to be translated. This command puts the address of the string header onto the stack. The characters are processed one by one and written back into the variable for return to BASIC.

Before BASIC can load the compiled program, it has to have some information about the file: type, segment, offset, and length. Type is a one-byte field, segment and offset are two-byte (word) addresses, and length is a two-byte field.

One quick and dirty way to pass this information to BASIC is to set up a seven-byte prologue at the beginning of the *CALL*ed program which contains the necessary information. Program length is determined by setting up an *EQU* statement to trap the starting address (BOF) and subtracting that address from EOF, which is defined at the program end. File type is defined as 0FDH which is BASIC's convention for a data file.

```
;Translate String to Upper Case
               XLAT      PROC FAR
               XLATO:    PUSH BP
                         MOV  BP,SP
                         MOV  SI,[BP+6] ;String header
                         MOV  CL,[SI]   ;String length
                         CMP  CL,0      ;Null string?
                         JZ   XLAT9     ;Yes - exit
                         XOR  CH,CH     ;Clear MSB
                         MOV  SI,[SI+1] ;String address
                         MOV  DI,SI
               XLAT3:    LODSB
                         CMP  AL,'a'    ;Lower case char?
                         JC   XLAT6     ;No
                         AND  AL,ODFH   ;Convert to upper case
               XLAT6:    STOSB          ;Replace in string
                         LOOP XLAT3
               XLAT9:    POP  BP
                         RET  2
               XLAT      ENDP
```

Listing 1.

The lines of prologue information presented in Listing 2 establish the segment, offset, and program length variables that will be stripped off by the *BLOAD* command. The *JMP* command at the bottom opens an entry point to the translate routine. (You would typically include several assembly language routines in one program, in which case there would be an additional *JMP* command for each routine.) An end of file marker (EOF DB 1AH) is written at the end of the program just prior to the *COMSEG ENDS* statement so that the program length can be calculated.

Because the assembler does not know that BASIC will strip off the seven-byte prologue when the program is loaded, you cannot use absolute jumps or addresses in the remainder of the program. All addressing must be relative, or the instruction pointer will be off by seven bytes and wind up in some nebulous never-never land, causing unpredictable results.

The subroutine code can now be assembled into a binary file and linked, after which it is ready to be loaded into memory. (Binary files are created from EXE files by running *EXE2BIN* against

LISPcraft
by Robert Wilensky

"[Wilensky's LISPcraft] offers a comprehensive tutorial in Franz LISP....It fills the vacuum created by the lack of books in Franz LISP....It can also be used as a textbook to learn other dialects of LISP ....It is well worth its price."

—Duvvuru Sriram,
Carnegie-Mellon University
From a review in
The Sigart Newsletter

Please send me _____ copies of LISPcraft by Robert Wilensky @$19.95 each/paperbound (NY and CA residents please add sales tax.)

Name ______________________________

Address ______________________________

____

City ____________ State ____________ Zip ____________

_____ Check enclosed _____ VISA _____ MasterCard

Account No. ____________ Exp. Date ____________

Norton

W. W. Norton & Company, Inc., 500 Fifth Avenue, NY, NY 10110

CIRCLE 69 ON READER SERVICE CARD

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. Detailed manual included. Dealer inquiries invited. WRITE is $239.00.

BDS's C Compiler

This is the compiler you need for learning the C language and for writing utilities and programs of all sizes and complexities. We offer version 1.5a, which comes with a symbolic debugger and example programs. Our price is (postpaid) $130.00.

Tandon Spare Parts Kits

One door latch included, only $32.50.
With two door latches $37.50.
Door latches sold separately for $7.00.

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, Osborne, KayPro, Otrona, Epson, Morrow, Lobo, Zenith, Xerox. Please request our new catalog. We welcome COD orders.

Workman & Associates
112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401

WORKMAN & ASSOCIATES
USER-FRIENDLY SOFTWARE

CIRCLE 68 ON READER SERVICE CARD

```
;Build file prologue for BASIC loader
                         DB   OFDH        ;File type
                         DW   OF77H       ;Default segment
                         DW   0           ;Default offset
                         DW   EOF-BOF     ;Program length
               BOF       EQU  $           ;Start of code
                         JMP  XLATO       ;Upper case translate
```

Listing 2.

the compiled code and specifying a .BIN extension for the output.) The actual load is done in the *CALL*ing BASIC program, using *BLOAD*, and depends on using *PEEK* to retrieve the correct loading address:

```
100 'Load machine language sub-
    routines
110 DEF SEG=0
120
    MLSEG
    =PEEK(&H510)
     +256*PEEK(&H511)+&H1001
130 DEF SEG=MLSEG
140 XLAT=0
150 BLOAD "SUBRTN.BIN",0
```

The success of this routine depends on the fact that DOS maintains the beginning segment address of BASIC's work area at hex 510-511. Adding hex 1000 (64K) to this address provides the ending address of BASIC in segment notation. An extra 16 bytes is then added to account for the memory management block that follows BASIC. The result of these calculations is an address in free memory above BASIC.

Notice that the entire program, SUBRTN.BIN, is loaded into memory. The translate routine, *XLAT*, is shown as entry point 0. If there were additional routines in the program they would be numbered with an offset of 3 for the byte length of the *JMP* command so that the second one would be equal to 3, the third to 6, and so on.

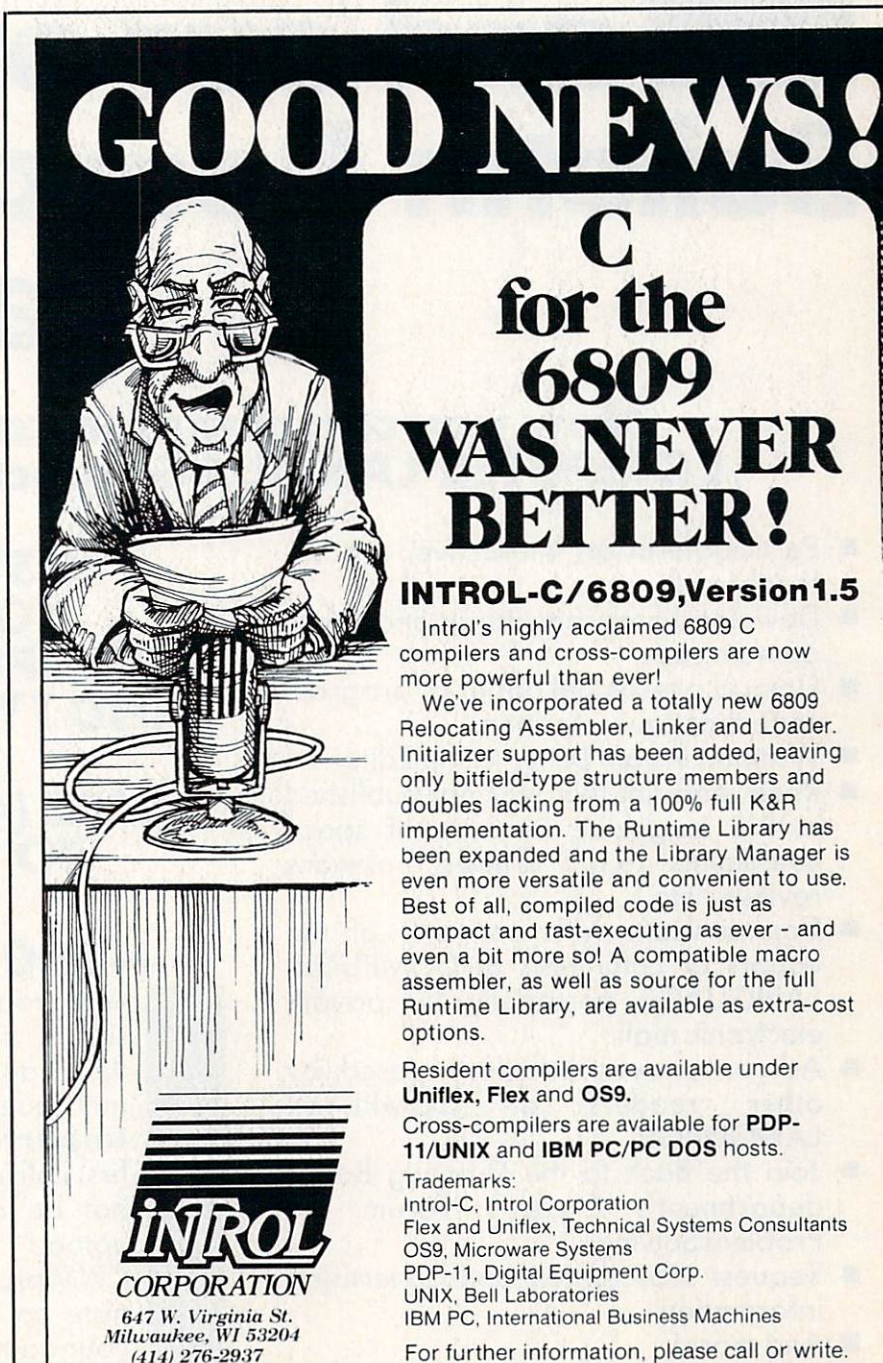
The absolute addressing approach

As mentioned previously, this system requires you to use relative addressing. A tidier solution is to write the machine language program as you normally would, omitting the load information, and then append those seven bytes to the beginning of the compiled and linked code. This scheme allows you to use absolute addressing since the seven load bytes are not present when the program is assembled.

I've written a routine called BASFMT.ASM to create the necessary load information for BASIC and insert it at the beginning of a binary (.BIN) assembly language program. If you'd like to obtain this rather long listing, I've placed in on the *COMPUTER LANGUAGE* Bulletin Board Service (415 957-9370) for you to download. You can also pick it up on *COMPUTER LANGUAGE*'s account on CompuServe. This program, however, must be compiled into an .EXE file and linked before it can be run.

Either of these schemes—adding a prologue and using relative addressing or inserting the load information ahead of the finished code—will let you load and call machine language subroutines via BASIC with a minimum of trouble. If you've been looking for a way to get faster response out of interpretive BASIC, try converting some of your common subroutines to assembly language and interfacing them with one of these methods. You'll be amazed at the difference in response time. ■

Jeri Girard, who has been a computer analyst for about 10 years, writes articles and reviews for several computer magazines. Her book, The Essential User's Guide to the IBM PC and PCjr, *published by Baen Enterprises, is due out this month.*



GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809,Version 1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

Trademarks:
Introl-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL CORPORATION
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

CIRCLE 32 ON READER SERVICE CARD

COMPUTER LANGUAGE now on CompuServe and our own Bulletin Board Service

Now you can communicate with COMPUTER LANGUAGE electronically!

- Participate in an interactive, reader-feedback forum
- Download program listings and public domain code
- Upload a technical paper or program code that you've written
- Write an instant Letter to the Editor
- Read material that was not published in the magazine because of space constraints (e.g., articles, software reviews, etc.)
- Communicate with any one of the editors or columnists at COMPUTER LANGUAGE personally by private electronic mail
- Ask or answer questions posed by other readers of COMPUTER LANGUAGE
- Join the Back to the Drawing Board department's Expert's Forum for Problem Solving
- Request subscription and advertising information
- And more!

JUST CALL INTO YOUR LOCAL COMPUSERVE NODE AND TYPE "GO CLM"!

or

CALL (415) 957-9370 TO REACH THE BBS

—300/1200 BAUD—

After you receive the "connect" signal, type ⟨return⟩ several times and answer "0" to the nulls question. Then, you're in! **Important Note:** On your first call into the system, you will not be able to download or upload files. Simply request CP/M privileges by ⟨e⟩ntering a note to "SYSOP" containing your name and address.

PUBLIC DOMAIN SOFTWARE REVIEW

By Tim Parker

One of the problems I encounter when writing this column is that with the wealth of material available it's very difficult to decide what to include and exclude each month. Also, the audience that this public domain code is intended for is diverse, as the readership of *COMPUTER LANGUAGE* varies from professional programmers to industry newcomers.

First I'd like to thank those people who've written me with their ideas on worthwhile public domain software. Your enthusiasm and criticism will add greatly to the effectiveness of this forum in the coming months.

And remember, the *COMPUTER LANGUAGE* Bulletin Board Service is the ideal medium for you to communicate with any of the columnists, including me! You can leave electronic messages to any user on the system, as well as download most of the code I've mentioned in this column. The BBS phone number is (415) 957-9370.

As promised in the last issue, we begin with a look at extended directory programs. The specific examples used in the next few paragraphs are for CP/M 80 (i.e., 8080) operating systems, although analogous programs exist for CP/M 86 and MS-DOS. (No, I have not forgotten all the sixteen biters: read on!) Usually the programs are called the same thing as in the CP/M 80 version to maintain a systematic catalog.

The directory programs are intended to replace the CP/M *DIR* command with a more powerful instruction set that also gives much more information. Using standard CP/M programs, the only method to get the sizes of all the files on a disk is with a *STAT* *.* command. *STAT*, of course, takes a little while to load, and the listing is in a vertical column. Hardly convenient if the disk has a lot of files.

To combine the two-program function, a number of public domain software releases are available. Many doing essentially the same task even have a variety of names. For example, DD.COM, SD.COM, D.COM, DIRR.COM, DF.COM, S.COM and XDIR.COM all display an alphabetically sorted listing of the files in a three- or four-column format, with the file size displayed next to each file name.

Each program has variations. Some display sizes as both records and kilobytes, some display the system attributes, some allow access to all user areas, and some allow the operator to set the display formats.

The most flexible of the group is XDIR.COM, which is part of Richard Conn's ZCPR (Z80 Command Processor Replacement) System (see Part I of his article on ZCPR3 in this issue and Part II in the November issue). Once this command processor replacement has been used, it becomes very difficult to return to CP/M.

XDIR.COM displays the name of a file, its size in kilobytes, the file attribute (Read/Only, Read/Write, or System), the sum of the file sizes displayed, the number of files on disk, the remaining space, and the drive/user area being checked. (It also has other features, such as allowing named directories, similar to UNIX, and logging files to the disk for scanning changes, which should be implemented with the ZCPR system.)

XDIR.COM allows a number of different display formats to be chosen by toggles appended to the *XDIR* command. The format of the *XDIR* command is *XDIR afn xxx* where *afn* is the ambiguous filename to be matched if required, and the *x*'s are optional toggles. (The default toggles can be set within the ZCPR program, so they don't have to be specified each time.) Help with the *XDIR* command is always available by typing *XDIR //* as the command.

File attributes to be displayed are toggled with an *An* command, where *n* is the attribute: *S* for system files, *N* for non-system files, and *A* for both. The *D* toggle sends output to the screen and a diskfile named XDIR.DIR, while a *P* will send it to the printer (allowing a comment line to be added). One usually overlooked use for the *XDIR P* command is the printing of labels for disk jackets that list the disk's contents. This is usually easier than using one of the available disk cataloging systems.

The *G* toggle allows grouping by either name-type or type-name. So an alphabetical listing by either filetype or filename is available. This is of immense value for language users, who can see all .REL files, .INT files, etc., grouped together, instead of scattered throughout the listing.

The *H* toggle changes from vertically to horizontally displayed alphabetic format (i.e., alphabetical displays either down the three columns or across them). Finally, the *N* toggle negates or complements the selected *afn* by displaying those files that do not match the ambiguous filename. Other toggles are available but most require ZCPR to be used, so will not be covered here.

XDIR.COM, while useful, is sometimes just too much information to be quickly digested when scanning disk directories. Also, it is a relatively large file, compared with others, such as DD.COM (11K vs. 3K). DD.COM, when invoked, displays a sorted directory in columnar format (sorted by filename and user number) showing file sizes and disk status. Switches can again be employed but must follow a dollar sign, as in *DD afn $x*, where *afn* is an ambiguous filename if required and the *x* is a toggled command option.

DD.COM supports four toggles. An *F* produces a fast listing with no file sizes displayed. An *S* lists those files with system attributes, while *V* produces a verbose listing giving file attributes and user numbers. Finally, a *U* lists files in all user areas (very handy).

As mentioned earlier, these programs have a number of variations, but XDIR.COM and DD.COM have been selected as representative of the two ends of the spectrum. XDIR.COM is available in several user disks, but the most recent version is XDIR3.COM, part of the second version of ZCPR. This program is on Volume 102 of the SIG/M disks (the .MAC file is on Volume 101) and on several other disks also. (ZCPR2 is on SIG/M Volumes 98 through 108, although not all are required.)

Those who are new to CP/M are invariably somewhat confused by the syntactical requirements of the system. Therefore, "front-end" programs that uncomplicate the various tasks are becoming popular. The public domain abounds with such programs, most of which do a very good job of their tasks. The problem with all of them is that for anyone who knows CP/M to even a minor degree, the programs tend to become frustrating as they are slow, somewhat redundant, and in the occasional case, imbecilic.

In general, such programs will display a list of the files on a disk and a menu that allows various files to be tagged for one of several functions, such as *TYPE*ing to the console, *ERA*seing, *REN*aming, or *PIP*ing to another disk. For those who do not want to learn the five basic commands of CP/M, I suppose that these programs will be of use. However, I have yet to meet a serious programmer who uses them. The time taken to load the programs and proceed through the sequences to get anything done always seems to be much longer than the brute-force method.

For those who are curious, however, a few such programs should be looked at. The two most frequently encountered programs are SWEEP.COM and WASH.COM, both of which have been through many versions. Another program called DISK.COM is still in its infancy, compared to the other two. Both SWEEP and WASH do boast impressive documentation and do their claimed tasks extremely well. One version of SWEEP (SWEEP37) is available on SIG/M Volume 110.

With these programs, mass renaming can be accomplished, such as *REN *.BAS=*.PAS*, which CP/M would respond to with an error. Also, files can be streamed for some function, such as printing out 10 files in a row. This can certainly be of use, however, I still maintain that these programs offer more frustration than comfort. Undoubtedly I'll get mail on that point! (Incidentally, ZCPR does all of this with no fuss at all . . . and it's the same price.)

Now, quickly, a look at a few of the more interesting recent releases from the public domain libraries. The addresses of the groups are listed at the end of this column.

For those with a curiosity about that artificial intelligence language LISP, there is a CP/M 86 version available on SIG/M Volume 153 and a CP/M 80 version on SIG/M Volume 118. Called XLISP: An Experimental Object Oriented Language,

CP/M-80 C Programmers . . .

Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast*, then it's time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming.

BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/Z80 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.

Plus . . .

- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner *and* the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group ($10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for it's remarkable *speed*.

BYTE Magazine placed BDS C ahead of all other 8080/Z80 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

"I recommend both the language and the implementation by BDS very highly."
Tim Pugh, Jr.
in *Infoworld*

"Performance: *Excellent.*
Documentation: *Excellent.*
East of Use: *Excellent.*"
InfoWorld
Software Report Card

". . . a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

Don't waste another minute on a slow language processor. Order your BDS C Compiler today!

Complete Package (two 8" SSDD disks, 181-page manual): **$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2 or higher. It is not currently available for CP/M-86 or MS-DOS.

BDSoftware

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

CIRCLE 5 ON READER SERVICE CARD

it implements a good number of the standard (if such a thing exists) LISP capabilities.

The system was originally written for CP/M 80 by David Betz, and the SIG/M disk contains the .COM file, a .DOC file, and a whole bunch of .C files. (Guess what language this one was written in!) A couple of assembly files and other useful material are included. The CP/M 86 version was modified by Harry Van Tassell and is essentially the same package. They both function identically as far as I have been able to tell.

The document file (29K) is readable, although only so much can be covered in such a file. Experimentation or supplemental reading will be of benefit to most users.

No serious bugs were found with either version, and they performed as they should have. This is definitely recommended for the curious and those who'd like to have a taste of another higher-level language. (No debates on that point, please.) The media as a whole seems to tout LISP as the closest thing possible to an artificial intelligence programming language, and for those who don't want to shell out the hundreds (or even thousands) of dollars for commercial versions, this is priced perfectly.

While on the subject of languages, probably the most controversial of all was released in a new version last year. Forth Version 8.3 (or Forth-83) is on SIG/M Volume 154. (Sorry, but I couldn't find a new version in either CP/M 86 or MS-DOS.) Forth-83 is supplied as a .COM file (no installation or assembly) occupying 24K. If memory serves correctly, this is less than the previous version, so whether a more efficient code has been used or some features have been dropped remains to be discovered.

A short .DOC file accompanies the .COM file, with a .HEX file of the kernel supplied. Squeezed files for assembler, debugging, multitasking (!), extension screens (!!) and direct BIOS I/O are included. Also, a CP/M interface is supplied as is a file labeled "Meta source and F83 screens".

Improvements over previous releases? Some of the less useful words of the earlier versions have been dropped or replaced with more powerful instructions. *DO* loops have been overhauled rather extensively and new words introduced. Many modifications have been made to the existing control words. Forth-83 attempts to come as close as possible to the proposed new Forth standard. I haven't had much opportunity to play with it, but it certainly looks interesting.

For those with a 68000, Forth 68000 is available on SIG/M Volume 151 along with several utilities for the DEC Rainbow, including a version of MODEM7.

A version of Small-C was released on SIG/M Volume 149 late last year for CP/M 86 implementations, along with an XMODEM written in FORTRAN for the VAX and one for the Zenith 100.

The Small-C is functionally identical to the CP/M 80 version and performs flawlessly. It is supplied as a .LBR file, taking 114K of space. A 1K documentation file is tacked on. Get a book on C to go with it.

To end the languages, the CBASIC Users Group Volumes 1 and 2 are available together on SIG/M Volume 163. This disk contains a potpourri of programs—

For your IBM/PC

mbp COBOL: 4 times faster, and now with SORT & CHAIN.

$750.

mbp COBOL can be summed up in one word: fast.

Because it generates native machine language object code, the mbp COBOL Compiler executes IBM/PC* programs *at least* 4 times faster (see chart).

GIBSON MIX Benchmark Results
Calculated S-Profile
(Representative COBOL statement mix)

Execution time ratio

mbp COBOL	Level II** COBOL	R-M*** COBOL	Microsoft**** COBOL
1.00	4.08	5.98	6.18

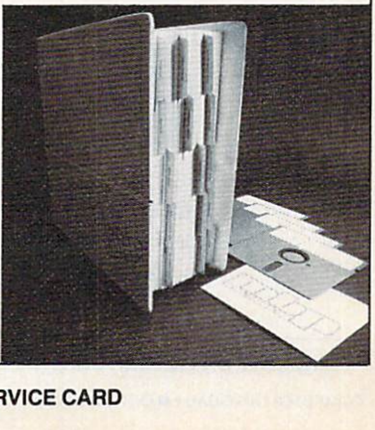
128K system with hard disk required. *IBM/PC is an IBM TM; **Level II is a Micro Focus TM; ***A Ryan-McFarland TM; ****A Microsoft TM.

Fast also describes our **new SORT,** which can sort four-thousand 128-byte records in less than 30 seconds. A callable subroutine or stand-alone, 9 SORT control fields can be specified. And our **new CHAIN** is both fast and secure, conveniently transferring control from one program to another, passing 255 parameters. Plus, **new extensions** to ACCEPT & DISPLAY verbs give better, faster interactive programming.

The complete COBOL. An Interactive Symbolic Debug Package included standard; Multi-Keyed ISAM Structure; listing options allow source & object code, map & cross-reference checking; GSA Certification to ANSI '74 Level II; mbp has it all.

It's no surprise companies like Bechtel, Chase, Citicorp, Connecticut Mutual, and Sikorsky choose mbp COBOL; make it your choice, too. mbp is available at Vanpak Software Centers, or direct. For complete information, write **mbp Software & Systems Technology, Inc.,** 7700 Edgewater Drive, Suite 360, Oakland, CA 94621, or phone 415/632-1555 —today.

mbp®



CIRCLE 39 ON READER SERVICE CARD

some useful, others not—for CBASIC users.

A representative sampling of the disk includes directory calls from CBASIC (useful), CBASIC data file creation routine (very useful), and a demonstration of the *GET* command (yawn). An *MEMTEST* function, console I/O capture, and sample data bases are also included. For those who use CBASIC as a primary language, this disk will have some interesting programs, although chances are the heavy-duty CBASIC users will have implemented the required functions themselves.

Finally, following up on the assemblers last month, a few cross assemblers are available for the Motorola family of processors. (Those mentioned function from CP/M 80 to the target.)

A 68000 and a 6800 cross assembler are on the same disk: SIG/M Volume 140. Another 68000 cross assembler is on SIG/M Volume 92, with a "Little ADA" set for a Polymorphic system. None of them have been tried by your intrepid reporter. Anyone wanna give me a Sage to try them? I'll even settle for a 68K board for a Compupro! As for a Polymorphic, it sounds more like a crustacean than a computer. (Just kidding, guys!)

And THAT IS THAT, at least for another month!

Coming up in the next issue: another language for you to play with (but I won't say which till then). Also, some general utilities of a useful variety.

Don't forget that you can quickly get a note to me if you want something dug up from my archives (a tottering pile of disks and paper on my desk), want to scream insults in my direction, or (heavens) say something nice through the *COMPUTER LANGUAGE* BBS. I check it every couple of days, so feel free to use it to its fullest. Naturally, there is always the post office. The editors forward everything to me, although anything that ticks will be submerged first.

Most of the programs mentioned in these columns will be placed online in the CP/M facility of the BBS. Even more programs will be available Real Soon Now (don't sue me, Jerry) on the *COMPUTER LANGUAGE* section of CompuServe. I'll be sysoping CompuServe for *COMPUTER LANGUAGE*, so check it out.

Till next time, remember: "Real Programmers Don't Use Read/Write Tabs!"

Useful addresses: SIG/M is at P.O. Box 2085, Clifton, N.J., 07015-2085. CP/MUG is at 1651 Third Ave., New York, N.Y., 10028. ■

Thunder Software

- **The THUNDER C Compiler** - Operates under the APPLE Pascal 1.1 operating system. Create fast native 6502 programs to run as stand alone programs or as subroutines to Pascal programs. A major subset of the C defined by K & R. Includes a 24 page users guide, newsletters. Macro preprocessor, runs on APPLE][,][+, //e, //c. Source code for libraries is included. **Only $49.95**
- **ASSYST: The Assembler System** - A complete 6502 editor/assembler and lister for APPLE DOS 3.3. Menu driven, excellent error trapping, 24 p. users guide, demo programs, source code for all programs! Great for beginners. **Only $23.50**
- **THUNDER XREF** - A cross reference utility for APPLE Pascal 1.1. XREF generates cross references for each procedure. Source code and documentation provided. **Only $19.95**

Thunder Software POB 31501 Houston Tx 77231 713-728-5501
Include $3.00 shipping. COD, VISA and MASTERCARD accepted

CIRCLE 65 ON READER SERVICE CARD

A general purpose programming language for string and list processing and all forms of non-numerical computation.

SNOBOL4+ —the entire SNOBOL4 language with its superb pattern-matching facilities • Strings over 32,000 bytes in length • Integer and floating point using 8087 or supplied emulator • ASCII, binary, sequential, and random-access I/O • Assembly Language interface • Compile new code during program execution • Create SAVE files • Program and data space up to 300K bytes RAM

Have you tried SNOBOL4+?

MUSIC ANALYSIS • TEXT PROCESSING • SYMBOLIC MATHEMATICS • ARTIFICIAL INTELLIGENCE
COMPILER PROTOTYPING • CRYPTOGRAPHY • LINGUISTICS • LIST PROCESSING • GAMES •

With **ELIZA** & over 100 sample programs and functions

For all 8086/88 PC/MS-DOS or CP/M-86 systems, 128K minimum 5¼" DSDD, specify DOS/CPM format

Send check, VISA, M/C to: **$95** plus $3 s/h

Catspaw, Inc.
P.O. Box 1123 • Salida, CO 81201 • 303/539-3884

CIRCLE 9 ON READER SERVICE CARD

microSUB:MATH

A library of **Numerical Methods Subroutines** for use with your **FORTRAN** programs.

Over sixty subroutines of:

FUNCTIONS
INTEGRATION
MATRICES
NON-LINEAR SYSTEMS
INTERPOLATION
LINEAR SYSTEMS
POLYNOMIALS
DIFFERENTIAL EQ

Versions now available for:

MS-DOS:	IBM	FORTRAN-77
	SuperSoft	FORTRAN IV
	Microsoft	MS-FORTRAN ver 3.2
	DRI	DR FORTRAN-77
CP/M-80:	Microsoft	F-80
	SuperSoft	FORTRAN IV

LICENSE, $250.
with SOURCE CODE, $600.
(Manual alone, $25.)

MATH
SUBROUTINE
LIBRARY

TRADEMARKS
Microsoft & MS: Microsoft Corp
CP/M & DR FORTRAN-77: Digital Research Corp
IBM: International Business Machines
SuperSoft: SuperSoft Corp

foehn consulting, PO Box 5123, Klamath Falls, OR 97601 503/884-3023

CIRCLE 23 ON READER SERVICE CARD

Elegance
Power
Speed

C Users' Group
Supporting All C Users
Box 287
Yates Center, KS 66783

CIRCLE 17 ON READER SERVICE CARD

eXchanger
CP/M ⟷ ISIS
Emulator

Now move files and programs between your CP/M-80 system and your Intel Series I or II MDS! The ICX package provides complete bidirectional file conversion capability, and even **allows execution** of ISIS-II programs under CP/M using the ISE emulator. The ICX Package is composed of the following two programs:

ICX A Deluxe bidirectional file conversion utility which works with your CP/M system and an 8" floppy drive to provide complete manipulation of an ISIS-II diskette. Takes directories, deletes files, and even initializes a blank disk with the ISIS file structure. Complete C source included. **$89**

ISE An ISIS-II Emulator which allows ISIS programs to run on any CP/M-80 system. Support for all ISIS-II system and monitor calls makes your CP/M micro look like an MDS! Supports banked memory. Complete MAC source included **$89**

Complete ICX Package (ICX & ISE) **$175**

Supplied on Single Density 8" Disk -
CP/M Digital Research, Inc. ISIS II Intel Corp

Western Wares
Box C
Norwood CO 81423
(303) 327-4898

CIRCLE 67 ON READER SERVICE CARD

EASY
To Use!

Developed in England by Southern Software

SBE

Z80 / 8088 (8086 / 80186 / 8087) machine-code development system. With latest Reduced Instruction-Set philosophy.

☐ SBE/TRS 80 (All DOS) **$100** + $3 s/h
☐ SBE/PC (PC-DOS/MS-DOS **$160** + $3 s/h

Allen Gelder Software
(415) 681-9371
Box 11721 San Francisco, CA 94101

CIRCLE 28 ON READER SERVICE CARD

EXOTIC LANGUAGE OF THE MONTH CLUB

PILOT: A specialized language for conversational scripts

By John A. Starkweather

PILOT is a widely used author language for the creation of computer-assisted instruction. It is a collection of specialized programming tools for the creation of computer dialogs.

PILOT is not an authoring system with pre-arranged formats ready for instructional use, though it can be used to create a variety of such formats. The language was designed to have an exceedingly simple entry level and immediate feedback for a novice program author who wishes to present information and evaluate a user's response. Because of this, it has been used as an introductory programming environment, especially successful with young children.

PILOT gets its name not only from the general goal of helping a novice become a computer pilot but also as a reminder of just some of the uses to which it has been put: the letters begin the words Programmed Inquiry Learning Or Teaching.

Because the PILOT system has been developed to be interactive and conversational, it allows the computer to play the role of a human helper. A PILOT program can act as though it is an individual tutor or a consultant. It can be the giver of tests and examinations, it can ask questions and prompt the collection of data, and it can describe and assist with the learning of other computer systems.

The features of PILOT can be combined with the advantages of other programming languages. Sometimes there are problems that require a good deal of explanation, questioning, and collection of data as a preliminary to extensive computation. PILOT can be combined with other systems to make it easy to program such an introductory section.

Major computational tasks can be turned over to other systems at the proper time after data has been collected. Improvement of the interactive portion of such a system is important, and PILOT makes it easier for the computer to prompt the user and explain what is desired. The user feels helped but still is in direct control.

While all this is possible with other computer languages, it is usually much more difficult and takes greater thought and effort on the part of the developer to program conversational interaction with a general purpose computer language.

Most conversations between two people involve an alternation of some simple basic elements. Each person needs to present information or ask questions. Each needs to listen and accept answers. As a third and more complex element, each person needs to evaluate what is seen or heard and respond differently as a result of that evaluation.

These three elements describe the major core mechanisms of PILOT. In order to remove any ambiguity about what the computer is expected to do, code letters followed by a colon are used to indicate when the computer is to perform one of these functions. These are called PILOT statements.

PILOT can present information or ask a question.

T: is followed by text (information or a question) that is to be typed or displayed by the computer. For example, the following line in a PILOT program:

```
T: Hello there, whoever you are.
```

will cause the computer to type or display:

```
Hello there, whoever you are.
```

Everything following the colon is displayed. There is no functional difference involved in arranging for the computer to ask a question. Thus, the following:

```
T:What is your name?
```

will cause the computer to type or display:

```
What is your name?
```

exactly copying the line of text that follows *T:*.

The *T:* (*Type*) statement is simply repeated in order to display more than one line of text. For example:

```
T: Two tractors can together plow a
T: field in eight hours.
T: How long will it take three tractors to
T: plow a field of the same size if all
T: tractors operate at the same speed?
```

will cause the computer to display

```
Two tractors can together plow a
field in eight hours.
How long will it take three tractors to
plow a field of the same size if all
tractors operate at the same speed?
```

PILOT can accept answers and try to find meaning.

A: directs the computer to accept an answer from the user. *M:* is followed by words or elements of text for which a match will be attempted within the text of the latest answer. For example:

```
A:
M: FRED
```

will cause the computer to accept an answer from the console keyboard and to look for a match with "FRED".

PILOT can react differently to different answers.

In order to take different actions in response to different answers, the letters *Y* or *N* may be added to the code letters in PILOT statements. These letters are conditioners that cause the statement to be effective or not depending upon the success of the last attempted match. If a match was found (YES), then a statement such as *TY:* will operate, while *TN:* will not. If a match was not found (NO), then the reverse will be true. In PILOT, such conditioners can be added to affect the operation of any statement.

Let's look at an example of PILOT coding for a conversational interchange. For the moment, assume that PILOT is in operation, and you have created and stored a brief PILOT program within the memory of the computer. PILOT displays the word "READY", indicating that it is waiting for a command. If you type the command *list*, PILOT will display the program. That means that you will see the sequence of PILOT instructions already prepared. If you type the command *run*, PILOT will run the program. That means that the computer will obey the sequence of instructions, and you will see the effect of their operation.

What the PILOT user types will be shown in bold face to distinguish it from what is typed by PILOT. Here is the program, as it would be displayed by the CP/M version called Nevada PILOT.

```
list
T:   Is it usually coider in the summer
 :   or in the winter?
A:
M:   winter
TY:  That's true most places.
TN:  You must be thinking of an unusual
   : place.
```

Here is the result of running the program:

```
run
     Is it usually colder in the summer
     or in the winter?
In the winter.
     That's true most places.

READY
```

We can run the program again and provide a different response:

```
run
     Is it usually colder in the summer
     or in the winter?
I'll say summer.
     You must be thinking of an unusual
     place.

READY
```

Let's review the listed program and see what caused the computer to respond as it did. A *T:* statement prompts the computer to type (display) whatever follows the colon. In this case it is a question. Next, the *A:* statement accepted an answer from the user. Following that, *M:* attempted to match the word "winter" with any portion of the response. If a match was found, then the response "That's true most places." was typed because *TY:* causes display only when a match has occurred. If a match is not found, then the alternate response occurred because *TN:* causes display only when a match has not occurred. The simple codes (*T:*, *A:*, and *M:*) at the front of each line indicate the basic conversational elements, and the extra letter *Y* or *N* conditions the operation according to whether or not a match has occurred.

1. Character set
   - A. Alphabetic characters (A-Z)
   - B. Numeric characters (0-9)
   - C. Special characters ("'()#$.,:;+-*/<>=?!@&)
2. Constants
   - A. Numeric constants—Core PILOT handles constants written as integers.
   - B. String constants—A sequence of alphabetic, numeric, or special characters.
3. General statement syntax

   ```
   <PILOT STATEMENT> ::=
   [<LABEL>] <INSTR> [<CONDITIONER>] [<RELATIONAL>] :
   [<OBJECT>]
   ```

   - A. *LABEL* (optional) consists of a name with the prefix *.
   - B. *INSTR* (required) consists of a single alphabetic character for core PILOT statements.
   - C. *CONDITIONER* (optional) is a single alphabetic character *Y* or *N* appended to the PILOT instruction. The conditioner causes a test of the results of the latest attempted match (the most recently executed *M* statement). If the conditioner is true, the current statement is executed. If the conditioner is false, the statement is skipped. The conditioner *Y* is true if the latest match was successful. The conditioner *N* is true if the latest match was unsuccessful.

     As an example, the statement *TY:HELLO* will display the word "HELLO" if an item of the last *M* statement matches with an element of the last input preceding it.
   - D. *RELATIONAL* (optional) is a conditional expression enclosed in parentheses and following the PILOT instruction (and any conditioner). If it is evaluated to be true, the statement is executed; otherwise the statement is skipped. The conditional expression is of the form:

     ```
     <numeric expression> <rel. op> <numeric expression>
     ```

     where *rel. op* (*relational operator*) refers to the symbols <, >, or = with the usual meaning for numeric expressions.

     Numeric expressions may be formed using the operators +, -, *, and /, interpreted as addition, subtraction, multiplication, and integer division respectively.

     As an example, the statement *T(A>B):HELLO* will display the word "HELLO" if the value of variable *A* is greater than the value of variable *B*. *T(X):HELLO* will display the word "HELLO" if the numeric value of *X* is greater than zero.
   - E. The colon (*:*) is required.
   - F. *OBJECT* comes after the colon, and its syntax depends upon the statement type.
4. Names and references
   - A. Labels begin with * and continue until the first blank. Labels must come first on a line and have at least one blank between the label and the code which may follow. The label may appear alone on a separate line, as in the following:

     ```
     *LABEL
     T:DISPLAY THIS
     ```

     Labels are used in the body of a *J* or *U* statement to refer to the place in the program sequence where the label occurs. Thus, *J:*LABEL* would cause a jump to the above sample program.

     Core PILOT handles labels of at least four characters.
   - B. Numeric variable names are single letters of the alphabet. They are preceded by # when in the context of a character string. When a numeric variable name appears in the body of an *A* statement, the input must be numeric. Its numeric value is stored and may be referenced by the name. Such references may appear in *T*, *Y*, or *N* statements and cause the number to be retrieved and displayed.

Table 1.

```
A:#X
T:THE VALUE IS #X
```

Numeric variables are automatically set to a value of zero when the program is started. They may also be defined and set to a value by the content of a *C* statement that contains an expression such as *Y=X-10*.

C. String variable names begin with *$*. When a string variable name appears in the body of an *A* statement, the text which is entered is stored and may be referenced by that name. Such references may appear in *T*, *Y*, or *N* statements and cause the text to be retrieved and displayed.

```
A:$NAME
T:HELLO, $NAME
```

In the above example, if "ROBERT" is entered at the *A* statement, then "HELLO, ROBERT" will be displayed.

A *T*, *Y*, or *N* statement containing a string name without previously entered text will display the name.

T:THIS IS $UNKNOWN will cause
"THIS IS $UNKNOWN" to be displayed

5. Core PILOT statement types

PILOT core statements are used in all versions of PILOT and are abbreviated to a single letter. They are:

T:	(Type)
A:	(Accept)
M:	(Match)
R:	(Remark)
J:	(Jump)
E:	(End)
U:	(Use)
C:	(Compute)

A. T (Type)

```
<T Object>          ::= [<T Argument List>]
<T Argument List>   ::= <T Argument> [<T Argument> ...]
<T Argument>        ::= <String Constant>
                    ::= <Numeric Variable>
                    ::= <String Variable>
```

The *T: (Type)* statement will display whatever is typed after the colon. If the *Type Object* is null (nothing after the colon), the statement produces an empty line. String constants are typed as is with no surrounding quotes. Variables are recognized by the variable prefix (*$* or *#*) in the object. Variable names are terminated by the first non-alphanumeric character following the prefix. When a variable appears in the *T Object* its name is replaced by its value. Items in the *T* Argument List are concatenated in the order they appear. For example:

```
T: HELLO $NAME, I UNDERSTAND YOU ARE #N YEARS OLD.
```

If *$NAME* = "FRED" and *#N* = 15, then the above statement would produce the following:

```
HELLO FRED, I UNDERSTAND YOU ARE 15 YEARS OLD.
```

Two additional forms of the *Type* statement are *Y* and *N*, which are exactly equivalent to *TY* and *TN* respectively. A colon by itself can be used for continuation lines.

B. A (Accept)

```
<A Object>      ::= <A Argument>
<A Argument>    ::= <Numeric Variable>
                ::= <String Variable>
```

The *A: (Accept)* statement is used to receive input from the keyboard. The response is edited, replacing any or no leading and trailing spaces with one space at each end and compressing multiple spaces between words into single spaces, and then kept in a buffer. If a variable name is present as an argument, the variable is set to the accepted value. In addition, many PILOTs provide a statement that can retrieve what is stored in a variable and place it in the accept buffer.

Table 1 (Continued).

ATTENTION: ENGINEERS PROGRAMMERS

PolyFORTH® II

the powerful multitasking/ multi-user operating system is now available for most micro-computers running—

CP/M*-80 and CP/M*-86

Offers CP/M users:

- **An ability to run multiple terminals**
- **Unlimited control tasks**
- **Concurrent printer operation**

These advanced features combine with FORTH, Inc.'s powerful version of the FORTH programming language to offer CP/M users the ideal environment for all interactive and real-time applications.

Featuring speed of operation, shortened development time, ease of implementation and overall cost-effective performance, this system is fully supported by FORTH, Inc.'s:

- **Extensive on-line documentation**
- **Complete set of manuals**
- **Programming courses**
- **The FORTH, Inc. hot line**
- **Expert contract programming and consulting services**

From FORTH, Inc., the inventors of FORTH, serving professional programmers for over a decade.

Also available for other popular mini and micro computers.

For more information contact:

FORTH, Inc.

2309 Pacific Coast Hwy.
Hermosa Beach,
CA 90254
213/372-8493
RCA TELEX: 275182
Eastern Sales Office
1300 N. 17th St. #1306,
Arlington, VA 22209
703/525-7778

*CP/M is a registered trademark of Digital Research

CIRCLE 24 ON READER SERVICE CARD

C. M (Match)

```
<M Object>          ::= <M Argument List>
<M Argument List>   ::= <M Argument>,<M Argument>, . . .
<M Argument>        ::= <String Constant>
                    ::= <String Variable>
```

The *Match* statement is used to compare a list of items with the contents of the accept buffer. For each argument, a scan is made of the accept buffer until a match occurs or the end of the string is reached. The match is successful if at least one argument was successfully matched. A match of any item causes a YES condition to be set, and if no item matches a NO condition is set. Following statements ending in *Y* are obeyed only if the YES condition is set and statements ending in *N* are obeyed only if the NO condition is set.

Arguments are separated by commas and may be string constants without quotes or string variables. Multiple spaces in match arguments are compressed to single spaces before matching, and leading and trailing spaces are significant. A comma that terminates the last item is ignored but can serve to indicate the presence of a trailing blank or blanks.

D. J (Jump)

```
<J Object>          ::= *<Label>
```

The *J*: (*Jump*) statement always has a label name after the colon and the name matches a label somewhere in the program. If the label is on a line by itself, execution continues with the first statement following the label. The *J Object* may be a label name with or without the label prefix (*).

E. C (Compute)

```
<C Object>          ::= <Assignment>
<Assignment>        ::= <Variable> = <Expression>
```

The *Compute* statement is used to assign values to numeric and string variables. The *C* statement was designed as one means of extending the language, and specific syntax was left undefined so that it might match that of an available general purpose language. Extensions of PILOT have in general followed BASIC-like syntax. Some have required prefixes to denote type for all variables and some have required this only for string variables, allowing statements such as *C:X=A+B*. The concatenation and assignment of string variables allows the development of responses that make use of elements of user interaction.

F. U (Use)

```
<U Object>          ::= *<Label>
```

The *Use* statement transfers control to the subroutine whose first statement is labeled with the *U Object*. The *Use* statement pushes the return location (the immediately following statement) onto a push-down stack of at least seven levels.

G. E (End)

The *End* statement is used to return from a subroutine or to terminate the PILOT program. The *E* statement pops the push-down stack, removing the return location from the stack.

H. R (Remark)

```
<R Object>          ::= <Anything>
```

The *Remark* statement is used for any comments that the author wishes to include in the program. The statement is ignored during execution. *R* statements may be labeled and may be the target of *J* or *U* statements.

I. : (Continuation of Type statements)

A colon at the beginning of any line will continue the object part of the preceding statement only if it was a *Type* statement (*T*, *Y*, or *N*). Labels, conditioners, and relationals are not allowed on continuation lines. If a conditioner or relational prevents execution of a statement, none of its continuation lines will be executed.

Table 1 (Continued).

As can be seen from these examples, PILOT relies on simple character string matching and does not have mechanisms for complex parsing of free-form input. It was designed as a practical tool of modest aims and not as a research device for artificial intelligence.

Knowing that PILOT would necessarily need changes from our first ideas, we decided to describe a required core to the language and indicate specific ways that extensions should be added. The features presented in Table 1 form a description of core PILOT as a minimal basis for the language, using standards developed by the major users of PILOT in 1973. Most versions of PILOT have additional features.

While core statement types were to use single letters, it was expected that the lan-

Cursor and screen controls:

CA:	Cursor address to set row and column
CH:	Clear and home
CL:	Clear to end of line
CE:	Clear to end of screen

Various other aids to conversation:

FOOT:	(Foot of screen halt and prompt)
PA:	(Pause)
VNEW:	(New Variables)
XI:	(Execute Immediate)
CALL:	(Call existing program)
XS:	(Execute from the System)

The *FOOT:* statement places a prompting line at the bottom of the screen and waits for a response before proceeding.

The *PAUSE:* statement halts program operation for a specified length of time, and then continues.

The *VNEW:* statement erases string or numeric variables.

The *XI:* statement obeys the contents of a string variable, which should be a valid PILOT statement. Such a string can be created as the result of prior interaction.

The *CALL:* statement provides a way to call upon the operation of a separate program that exists in the computer memory external to PILOT.

The *XS:* statement calls upon the computer operating system to initiate operation of another program. This can be a useful linkage in using PILOT for an interactive front end to other operations.

Table 2.

guage would be extended by the addition of other multi-letter names for new functions. This would indicate that they might not be the same in different versions of PILOT. Some common additions are presented in Table 2.

Most versions of PILOT have file management and data collection facilities with statement names that relate to the operating system in use. Recent versions have graphic facilities that depend on specific hardware.

The PILOT language developed as a result of experience with an interactive computer system called COMPUTEST[1], developed at the Univ. of California in San Francisco, Calif., and in a local elementary school district in 1962.

This system ran on a small IBM 1620 computer and was able to carry out interactive programs with one user at a time by means of its console typewriter. In early versions the COMPUTEST system operated with program information (the program written in COMPUTEST) stored in a deck of punched cards and read into the computer a few at a time when the program was executed. Later, extended versions of the system had greatly increased capabilities by using disk storage units that allowed random access to the program material.

When an IBM 360 model 50 computer with time-sharing capability became available at the UCSF campus, a new computer-assisted instruction system called PILOT[2] was developed. The language features of the PILOT system were patterned after those of COMPUTEST and stressed easy entry of instructional material by the program author.

It was felt that a teacher should not have to become a computer expert in order to develop a computer-assisted instructional program. The PILOT system began to be used for traditional frame-oriented instructional programs as well as providing practice with simulated clinical situations, self-evaluation testing, and simulated interviewing[3,4] with natural language input from the students.

Not long after the original version of the PILOT language was written for the IBM 360 computer, several versions of the language were programmed to run on other computers, some of them translating PILOT to a more general purpose language such as BASIC, SNOBOL, or an assembly language specific to the computer. Some were called PILOT, and some had derivative names such as PYLON and NYLON.

Partly because of the requirements of different computer systems and partly because of individual preferences, each version of the language was somewhat different from the others. In early 1973, representatives of six of the major versions of the PILOT language met together and agreed on a set of core language specifications to be common among all the systems. This language represented the experience of many workers in computer-assisted instruction at that time.

Recognizing that differences would undoubtedly arise, there was also agreement on a standard means of describing functions outside the core language. Because many of those involved had applications in elementary and secondary school settings, the resulting standard, called PILOT 73, was intended to be a language that would be easy for the program author to learn.

Although this was prior to the availability of microcomputers, the Datapoint 1200 desktop computer provided a means to investigate self-contained operation and avoid the problems of communication with early time-sharing systems[5].

This machine had a built-in minicomputer, two magnetic cassette tape drives, a keyboard, and a CRT for character display. The memory capacity of the first of these machines was 8,000 bytes. About three-quarters were used for the PILOT system, which left room for about 2,000 characters of program material at a time.

This was enough to hold a number of instructional frames in active memory and additional material was read from tape when it was needed. The tape could contain about 100,000 characters of program material and was sufficient for the development of modest adjunct elements of interactive instruction.

A major dialect of PILOT that departed from some of the PILOT 73 guidelines for core PILOT was developed by George Gerhold and Larry Kheriaty at Western Washington State College in Bellingham, Wash. It is called COMMON PILOT and has extensions and additional functions to handle more complex programming needed in college level instruction, particularly in mathematics and science. This version of PILOT has formed the basis of PILOT for the Apple microcomputer[6], for Radio Shack microcomputers[7], for C-Pilot on UNIX systems[8], and for PC/PILOT for the IBM Personal Computer[9].

PILOT 73 for the Datapoint computer became the basis of versions for an early microcomputer, the Processor Technology Sol. Before the company went out of business, it published a cassette version of PILOT and a version for its Helios disk system was in preparation. These were distributed by PROTEUS (The Processor Technology Users Group)[10].

ATTENTION: ENGINEERS PROGRAMMERS

PolyFORTH® II

the operating system and programming language for real-time applications involving ROBOTICS, INSTRUMENTATION, PROCESS CONTROL, GRAPHICS and more, is now available for . . .

IBM PC*

PolyFORTH II offers IBM PC users:

- **Unlimited control tasks**
- **Multi-user capability**
- **8087 mathematics co-processor support**
- **Reduced application development time**
- **High speed interrupt handling**

Now included at no extra cost: Extensive interactive **GRAPHICS SOFTWARE PACKAGE!** Reputed to be the fastest graphic package and the only one to run in a true multi-tasking environment , it offers point and line plotting, graphics shape primitives and interactive cursor control.

PolyFORTH II is fully supported by FORTH, Inc.'s:

- **Extensive on-line documentation**
- **Complete set of manuals**
- **Programming courses**
- **The FORTH, Inc. hot line**
- **Expert contract programming and consulting services**

From FORTH, Inc., the inventors of FORTH, serving professional programmers for over a decade.

Also available for other popular mini and micro computers.

For more information contact:

FORTH, Inc.

2309 Pacific Coast Hwy.
Hermosa Beach,
CA 90254
213/372-8493
RCA TELEX: 275182

Eastern Sales Office
1300 N. 17th St.
Arlington, VA 22209
703/525-7778

*IBM PC is a registered trademark of International Business Machines Corp.

CIRCLE 25 ON READER SERVICE CARD

The Preferred C Compiler

"...C86 was the only compiler we tested that ran every benchmark we tried and gave the expected results... Computer Innovations C86 was the compiler that our staff programmers used both before and six months after we conducted the tests."
J. Houston, BYTE MAGAZINE - February 1984

*FAST EXECUTION -
of your programs.
*FULL & STANDARD IMPLEMENTATION OF C -
includes all the features described by K & R. It works with the standard MSDOS Linker and Assembler; many programs written under UNIX can often be compiled with no changes.
*8087 IN-LINE -
highly optimized code provides 8087 performance about as fast as possible.
*POWERFUL OPTIONS -
include DOS2 and DOS1 support and interfaces; graphics interface capability; object code; and librarian.
*FULL LIBRARY WITH SOURCE -
6 source libraries with full source code the "large" and "small" models, software and 8087 floating point, DOS2 and DOSALL.
*FULL RANGE OF SUPPORT PRODUCTS FROM COMPUTER INNOVATIONS -
including Halo Graphics, Phact File Management, Panel Screen Management, C Helper Utilities and our newest C_to_dBase development tool.
*HIGH RELIABILITY -
time proven through thousands of users.
*DIRECT TECHNICAL SUPPORT -
from 9 a.m. to 6 p.m.

Join The Professional Programmers Who Agree C86™ Is The C Compiler Of Choice

For Further Information Or To Order Call:
800-922-0169
Technical Support: (201) 542-5920

980 Shrewsbury Avenue
Suite PW509
Tinton Falls, NJ 07724

Computer Innovations, Inc.

C86™

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
UNIX IS A TRADEMARK OF BELL LABS. C86 IS A TRADEMARK OF COMPUTER INNOVATIONS, INC. MSDOS IS A TRADEMARK OF MICROSOFT.
PCDOS IS A TRADEMARK OF INTERNATIONAL BUSINESS MACHINES

CIRCLE 13 ON READER SERVICE CARD

Atari Inc. has developed PILOT for its microcomputer in a version that is easy for elementary students to use[11]. It contains graphic extensions providing the kind of "turtle" graphics found in LOGO. Although PILOT itself does not use line numbers, the Atari version requires them for program entry and editing, using the same mechanisms as provided for BASIC.

PILOT for microcomputers using the CP/M operating system was developed to provide the standard version for users of a wide range of equipment having disk storage. It is made available under the name Nevada PILOT by Ellis Computing in San Francisco, Calif.[12] This version contains a built-in screen editor that can be configured for operation with many different terminals.

In addition to its primary use for computer-based instruction or testing, PILOT has proven to be a useful tool for the rapid development of interactive menus that are sometimes useful for data entry or in assisting users with the operation of complex programs. When applied to instruction and training, the addition of graphics seems quite necessary—particularly a rapid line drawing capability to produce the equivalent of a teacher's blackboard diagrams.

One can, and should, write PILOT programs in a modular, top-down fashion, but the global nature of all variables is sometimes a limitation in extensive programs. An ability to pass arguments to subroutines having local variables could be useful, but it would not assist the novice user for whom the language was initially designed. ■

References

1. Starkweather, J.A. "Computest: A Computer Language for Individualized Testing, Instruction, and Interviewing." *Psychological Reports,* 1965, 17:227.
2. Starkweather, J.A. "A Common Language for a Variety of Conversational Programming Needs." *Readings in Computer-Assisted Instruction,* H.A. Wilson and R.C. Atkinson (Editors), New York: Academic Press, 1969, p. 269.
3. Starkweather, J.A., Kamp, M., and Monto, A. "Psychiatric Interview Simulation by Computer." *Methods of Information in Medicine,* 1967, 6:15.
4. Kamp, M. "Evaluating the Operation of Interactive Free-Response Computer Programs." *Journal of Biomedical Systems,* 1971, 2:33.
5. Kamp, M. and Starkweather, J.A. "A Return to a Dedicated Machine for Computer-Assisted Instruction." *Comp. Biol. Med.,* 1973, 3:293.
6. *Apple Pilot* and *Apple Super Pilot,* Apple Computer Inc. 10260 Bandley Dr., Cupertino, Calif. 94017.
7. *TRS-80 Pilot,* Radio Shack Education Div. 1600 One Tandy Center, Fort Worth, Texas 76102.
8. Sumner, T. *C-Pilot Language Reference Manual,* Alamonville, Ltd., P.O. Box 27186, Concord, Calif. 94527.

9. *PC/PILOT Language Reference Manual,* Washington Computer Services, 3028 Silvern Lane, Bellingham, Wash. 98226.
10. *Processor Technology Pilot,* PROTEUS, 1690 Woodside Road, Suite 219, Redwood City, Calif. 94061.
11. *Atari Pilot,* Atari Inc. 1272 Borregas Avenue, Sunnyvale, Calif. 94086.
12. *Nevada Pilot,* Ellis Computing, Inc. 3917 Noriega Street, San Francisco, Calif. 94122.

John Starkweather and his colleagues at the Univ. of California at San Francisco originated the PILOT language. Starkweather is a professor of medical psychology in the Univ. of California School of Medicine and a former director of the Computer Center at UCSF. Portions of this article are adapted from A User's Guide to PILOT, *to be published by Prentice-Hall.*

The C Interpreter:
Instant-C™

Programming in C has never been Faster.
Learning C will never be Easier.

Instant-C™ is an optimizing interpreter for the C language that can make programming in C three or more times faster than when using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use and learn as Basic. Yet ***Instant-C***™ is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into ***Instant-C***™ for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only once.

Instant Execution. ***Instant-C***™ uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

Instant Loading. Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Get more done, faster, with better results. ***Instant-C***™ is available now, and works under PC-DOS*, MS-DOS*, and CP/M-86*.

Find out how ***Instant-C***™ is changing the way that programming is done. ***Instant-C***™ is $500. Call or write for more information.

Rational
Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), PC-DOS (IBM), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

CIRCLE 56 ON READER SERVICE CARD

ATTENTION:
ENGINEERS
PROGRAMMERS

PolyFORTH® II

the operating system and programming language for real-time applications involving ROBOTICS, INSTRUMENTATION, PROCESS CONTROL, GRAPHICS and more, is now available for...

DEC* PDP-II*
and
LSI-II* Systems

The PolyFORTH II high performance features include:

- **Multiple users (30 terminals on a LSI-II)**
- **Unlimited control tasks**
- **High speed interrupt handling**
- **Reduced application development time**

PolyFORTH II software will run on any standard **PDP* or LSI-II** with RX02 disk (RSX* optional), **Micro/PDP-II*** and **PROFESSIONAL* 350** and is fully supported by FORTH, Inc.'s:

- **Extensive on-line documentation**
- **Complete set of manuals**
- **Programming courses**
- **The FORTH, Inc. hot line**
- **Expert contract programming and consulting services**

From FORTH, Inc., the inventors of FORTH, serving professional programmers for over a decade.

Also available for other popular mini and micro computers.

For more information contact:

FORTH, Inc.

2309 Pacific Coast Hwy.
Hermosa Beach,
CA 90254
213/372-8493
RCA TELEX: 275182

Eastern Sales Office
1300 N. 17th St.
Arlington, VA 22209
703/525-7778

*Registered trademarks of Digital Equipment Corp.

CIRCLE 26 ON READER SERVICE CARD

MicroMotion

MasterFORTH

It's here — the next generation of MicroMotion Forth.

- Meets all provisions, extensions and experimental proposals of the FORTH-83 International Standard.
- Uses the host operating system file structure (APPLE DOS 3.3 & CP/M 2.x).
- Built-in micro-assembler with numeric local labels.
- A full screen editor is provided which includes 16 x 64 format, can push & pop more than one line, user definable controls, upper/lower case keyboard entry, A COPY utility moves screens within & between lines, line stack, redefinable control keys, and search & replace commands.
- Includes all file primitives described in Kernigan and Plauger's Software Tools.
- The input and output streams are fully redirectable.
- The editor, assembler and screen copy utilities are provided as relocatable object modules. They are brought into the dictionary on demand and may be released with a single command.
- Many key nucleus commands are vectored. Error handling, number parsing, keyboard translation and so on can be redefined as needed by user programs. They are automatically returned to their previous definitions when the program is forgotten.
- The string-handling package is the finest and most complete available.
- A listing of the nucleus is provided as part of the documentation.
- The language implementation exactly matches the one described in FORTH TOOLS, by Anderson & Tracy. This 200 page tutorial and reference manual is included with MasterFORTH.
- Floating Point & HIRES options available.
- Available for APPLE II/II+/IIe & CP/M 2.x users.
- MasterFORTH – $100.00. FP & HIRES – $40.00 each
- Publications
  - FORTH TOOLS – $20.00
  - 83 International Standard – $15.00
  - FORTH-83 Source Listing 6502, 8080, 8086 – $20.00 each.

Contact:

MicroMotion
12077 Wilshire Blvd., Ste. 506
Los Angeles, CA 90025
(213) 821-4340

CIRCLE 40 ON READER SERVICE CARD

Fortran Scientific Subroutine Package

Contains Approx. 100 Fortran Subroutines Covering:

1. Matrix Storage and Operations
2. Correlation and Regression
3. Design Analysis
4. Discriminant Analysis
5. Factor Analysis
6. Eigen Analysis
7. Time Series
8. Nonparametric Statistics
9. Distribution Functions
10. Linear Analysis
11. Polynomial Solutions
12. Data Screening

Sources Included. Microsoft 3.2 compatible.

$295.00

FORLIB-PLUS™

Contains three assembly coded LIBRARIES plus support, FORTRAN coded subroutines and DEMO programs.

The three LIBRARIES contain support for GRAPHICS, COMMUNICATION, and FILE HANDLING/DISK SUPPORT. An additional feature within the graphics library is the capability of one fortran program calling another and passing data to it. Within the communication library, there are routines which will permit interrupt driven, buffered data to be received. With this capability, 9600 BAUD communication is possible. The file handling library contains all the required software to be DOS 3.0 PATHNAME compatible.

$69.95

Strings & Things ™

Character Manipulation and Much More!

$69.95

ALPHA COMPUTER SERVICE

P.O. Bos 2517
Cypress,CA 90630 **(714) 894-6808**

California residents, please add 6% sales tax

CIRCLE 2 ON READER SERVICE CARD

DESIGNER SCREENS

"A 100 to 1 Productivity Increase Over Coding"

Provides full-screen editing of terminal screen design images. And, a linker that generates self-relocating, 8080 machine language, run-time support.

Makes it easy to implement on-screen forms, menus, help screens, boiler-plate notices, and even simple animation.

Run-time support for input includes: data type control, decimal alignment, a type ahead buffer, end-user edit commands, and everybody's favorite, "Fred's Magic Window."

Fred's Magic Window can display field-by-field input instructions as needed, automatically.

Can be used with any computer language that allows programmed calls to CP/M 2.2. Great with assembly language or BDS C.

Runs on 80 × 24 or larger ASCII terminals. Supports five display attributes and line drawing. Designs are transportable between installed terminals.

Manual only:	**$ 10.00**	**(Check it out!)**
Software:	**185.00**	**(Supplied on: 8" SSSD CP/M or call.)**
Complete:	**$195.00**	

(Calif. residents add sales tax)

Austin E. Bryant Consulting
P.O. Box 1382, Lafayette, CA 94549
(415) 945-7911

CP/M is a trade mark of Digital Research
BDS C is a trade mark of BD Software

CIRCLE 7 ON READER SERVICE CARD

THE CODE SWAP SHOP

Editor's Note: All programs referred to in this reader-inspired, public domain column will always be available for downloading when you call the COMPUTER LANGUAGE *Bulletin Board Service at (415) 957-9370—300/1200 baud—or when you dial into CompuServe and invoke our account by typing "GO CLM".*

[redacted]

A complete data base management system

For the price of a telephone call to our bulletin board computer or to our account on CompuServe, you can download a fast, flexible, easy-to-use, and transportable data base management program called **The Creator**. Written in Microsoft BASIC by Bruce Tonkin of Round Lake, Ill., this DBM program can manage over 10,000 records and is completely expandable.

Now in its 10th version since 1980, The Creator will not require that sorting be done when you want to do simple data entry, update, or retrieval; consequently, unless and until you want sorted reports, it avoids wasted time incurred while waiting for the computer to sort or index your data file before you begin entering data.

Tonkin claims the product is "bug-free, fully supported, and completely documented." A 100-page manual featuring a full table of contents, a complete index, and also containing examples, hints, and explanations, can be obtained by sending to T.N.T. Software Inc., 34069 Hainesville Rd., Round Lake, Ill. 60073.

[redacted]

A calendar program written in Forth

Would you like to have a program that prints out any month in any year and also displays an attractive Rorschach-type pattern at the top of every month?

Adaptable to any machine running Forth, this program is not only attractive and useful, it's a good example of a cleanly written Forth program, and it's FREE. Well, almost. You have to call into our BBS or into CompuServe to get it. The author, Anthony T. Scarpelli of Portland, Maine, has offered it up to the public domain for us all to play with.

[redacted]

A text file listing program in C

How many times have you used TYPE to display a text file on your screen, only to have the display go scrolling off the screen before you could stop it with a Control-S?

A program called **LIST** has been written for those people without a TYPE-like program that has a screen pause feature built in. Written with DeSmet C for MS-DOS, this text file listing program will display a file sequentially and pause between every 23 lines. At each pause, LIST will display "Strike any key to continue . . ." and wait for your input. Like TYPE, when a Control-C is entered, the program is aborted.

LIST was contributed for use by the readers of *COMPUTER LANGUAGE* by Michael D. O'Quin from Medford, Ore.

[redacted]

Control your home with a timing program

Want a program that controls the "things" (appliances, lights, etc.) inside your house? Dennis Cashton of Wantagh, N.Y., has written a **Home Control Program** for his Heath BSR X-10 computer. If you look at the way Cashton designed the code for his system, you might be able to extract the ideas from his source code listing for a similar project of your own.

[redacted]

Z-80 assembly language debugging utilities

John P. Comiskey of Staten Island, N.Y., offers the readers of *COMPUTER LANGUAGE* two Z-80 assembly programs which, when called as subroutines by a Z-80 program, display the contents of the active registers, flag register, stack pointer, registers IX and IY, and the program counter on the screen and on the printer.

Written on the TRS-80 Model III, the programs are loaded into memory with the (TRSDOS) *LOAD* command and called with the *CALL* command. Each program begins by saving register A, then saving registers H and L, and popping the stack into HL to obtain the program counter. The programs then print headings and the contents of the registers before returning to the calling program.

[redacted]

A guessing game program that cheats!

OK, here's something you can't overlook. Contributed by Anton Dovydaitis of Santa Cruz, Calif., this program is simply called **Marvin**. The author did provide a subtitle, though—"a user-abusive guessing game."

If you're in the mood to download a program that may insult you . . .

The author wrote the program to illustrate a programming philosophy he calls "disciplined laziness." Dovydaitis uses Marvin to exemplify a well-written, top-down, structured programming style. "Programming is an art," he says, "and should reflect the artist." As an example of structured style, while playing with (or maybe, against) Marvin, puns actually have practical value in a program when written as mnemonics.

[redacted]

Do you have code for the Swap Shop?

If you've written a program that you'd like to see distributed free of charge to *COMPUTER LANGUAGE* readers, send us a two- to four-paragraph summary of what the program does, how you can make the program electronically available to our magazine (e.g., bulletin board transfer, disk format, CompuServe, etc.), and whether you'd like your name, address, and/or telephone number included in the magazine.

Address all correspondence to: Craig LaGrow, Editor, 131 Townsend St., San Francisco, Calif. 94107. Or call us up on CompuServe or the BBS!

SOFTWARE REVIEWS

Product reviews on CompuServe and the *COMPUTER LANGUAGE* Bulletin Board Service

mbp COBOL
DRI's DR FORTRAN-77
Systems Guild's CGRAPH
Creative Solutions's MacForth
Waltz Lisp
Nevada COBOL
Bendorf's Professional Programming Environment
Hendrix's Small-Tools
Borland's SideKick
Bellesoft's ESP
System/Z's BASIC/Z
Trio System's C-INDEX+

Telesoft Ada

Hardware Requirements: IBM PC or PC/XT with a minimum of 256K (8087 numeric co-processor chip optional required for floating point applications); DEC VAX (under VMS or UNIX); IBM 370 (under MVS or CMS); or, Motorola 68000-based systems (UNIX or Telesoft's proprietary ROS)
Price: $1,200 (IBM PC and PC/XT), $3,400 (68000 UNIX), $4,435 (68000 ROS), $10,905 (VAX UNIX and VMS); $11,065 (IBM 370 MVS and CMS)
Available from: Telesoft, 10639 Roselle St., San Diego, Calif. 92121, (619) 457-2700
Support: 3-month support free, annual maintenance fee of $150 for the IBM PC and PC/XT

The U.S. Department of Defense (DOD) is the driving force behind the development of the Ada language. Last year an Ada '83 standard was established, upon which commercial compilers will be validated.

Telesoft is among the first companies to produce a full Ada for minicomputers. This company has also created an Ada language subset for the IBM XT. The IBM subset version is the product reviewed here.

The package comes with 12 diskettes and a huge three-ring binder manual. Inside, the manual contains sections on the following:

- Run-time operating system (ROS) shell commands
- Ada utility packages to enhance the language
- Filer manual
- Text editor manual
- MC68000 macro assembler manual

- Filer manual
- Text editor manual
- MC68000 macro assembler manual

- Native-code linker user's manual
- Run-time kernel manual (MC68000 version)
- Ada compiler user's manual
- Pascal compiler user's manual
- Module and package composer user's manual
- Installation and operator's guide for Telesoft software on the IBM PC/XT.

The Telesoft package comes with its operating system—the run-time operating system (ROS). The latter resembles earlier versions of the UCSD p-system. This is not a coincidence since Dr. Kenneth L. Bowles, president of Telesoft, had previously organized the UCSD Pascal project in 1974.

The operating system differs from the UCSD p-system by the absence of a rigid main menu. Instead, ROS prompts the user with the ">" symbol. This allows the user to choose from the many shell commands available. In addition, on-line help is available. The user first encounters a general help menu from which a topic can be selected. When a user needs help on a specific subject, the display will show a copy of those manual pages that pertain to the area of difficulty.

The advantage in having shell commands is the flexibility gained by the user. The majority of these commands allow switches, file names, and other options that greatly enhance the power and versatility of the system. The available commands offer a variety of functions and utilities. A list of an example selection is:

- Invoking the Telesoft Ada and Pascal compilers
- Assigning volumes to unit numbers—very similar to the UCSD p-system
- Filtering standard input to standard output
- Date inspection and setting
- Creating and defining shorthand command line tokens
- Comparing different files
- Dumping a file's contents in hexadecimal
- Invoking the interactive screen editor (which is the same editor as in the earlier versions of the UCSD Pascal compiler packages)
- Invoking the file system manipulations (similar to the filer in early versions of the UCSD Pascal compiler packages)
- Searching for a character pattern in a text file
- Invoking the form document generator used to produce letters and documents
- Listing file and volume information
- Inspecting a file's content on the screen, one page at a time
- Printing the content of a text file
- Invoking the text formatter and printing utility program
- Removing one or more files from a directory
- Running a pseudo-code program
- Non-destructive disk scanning
- Sorting the contents of textual files on a line oriented basis
- Transferring files by replication (wildcards are allowed following the same UCSD convention)
- Remote communication
- UNIX-like piping capabilities.

The reader who is familiar with the UCSD p-system will notice that some of the above commands are available as an option presented by the UCSD filer. The same is true about the Telesoft filer. It seems the operating system designers at Telesoft chose to allow for the user to also access these utilities from the shell directly.

Installing Telesoft ROS was quite intriguing. Initially I expected ROS to reformat my already PC-DOS-formatted hard disk—as is done in the UCSD p-system. It didn't! ROS simply overwrote my PC-DOS files and established its own directory. This was done via *script* files (or *batch* files as they are known to PC-DOS users). These script files make use of a Pascal-like shell language that uses *IF-THEN-ELSE*, *GOTO*s, and labels to control the sequence of events.

I had few problems with the script file operations. At one stage, while trying to copy an Ada demonstration program to the hard disk, something went wrong and I found myself back to earlier stages, which forced me to transfer the ROS Ada files again. To solve the problem, I had to decline transferring the demo programs, as the script file prompted me. I used the transfer utility from the shell to successfully copy the demo and other programs to the hard disk.

The Telesoft interactive editor is identical to the one with Apple UCSD Pascal (version 1.1). I had no problem using the editor except that I currently happen to use more powerful ones that run under PC-DOS. Telesoft's editor is a typical screen-oriented program that possesses the ability to add, insert, delete, and change text, as well as the ability to find and replace text.

The editor uses the top line to display a menu and the current mode you are in. To go from one mode to another you must press Control-C, which initially puts you back in the main editing menu. You can then move into another mode (e.g., deletion, change, etc.). This can become frustratingly slow after some time working with the system, though.

The Telesoft filer is easy to use because it is modeled after the Apple UCSD Pascal filer. This incorporates file transfer, removal, name change, on-line volumes listing, file listings from within specific volumes, date alteration, and the ability to initiate a volume that will also check for bad blocks. Telesoft has not incorporated the enhancements that are available with version IV.1 and later—such as the capability of creating sub-volumes (i.e. sub-directories) as well as mounting and dismounting them. I found it easier to transfer files by using the "T" utility program from the shell.

As mentioned earlier the Telesoft Ada compiler for the IBM PC/XT is only a *subset* of the Ada language. The manual devotes 14 pages to those unimplemented features in Ada's full syntax and to the restrictions placed upon the user for these features being absent. Among the missing features are:

- The floating point patch does not implement mathematical functions like *SIN*, *COS*, *TAN*, and *EXP*
- Only the LIST and SUPRESS programs are supported
- Derived types
- Floating point attributes
- Fixed-point types
- Dynamic, open arrays.

The Telesoft Ada package, however, does include a library of utility subroutines like:

- Direct I/O package for random-access file manipulation
- Directory access from Ada programs
- Heap manager package
- Minimum/maximum package for integers, floats, characters and strings
- Cursor control package
- Sequential I/O package for general file manipulation
- Text_Io package (a very important

Eco-C Compiler
Release 3.0

We think Rel. 3.0 of the Eco-C Compiler is the fastest full C available for the Z80 environment. Consider the evidence:

Benchmarks*
(Seconds)

Benchmark	**Eco-C**	Aztec	Q/C
Seive	29	33	40
Fib	75	125	99
Deref	19	CNC	31
Matmult	42	115	N/A

**Times courtesy of Dr. David Clark*
CNC - Could Not Compile
N/A - Does not support floating point

We've also expanded the library (120 functions), the user's manual and compile-time switches (including multiple non-fatal error messages). The price is still $250.00 and includes Microsoft's MACRO 80. As an option, we will supply Eco-C with the SLR Systems assembler - linker - librarian for $295.00 (up to six times faster than MACRO 80).

NEW RELEASE

For additional information, call or write:

ECOSOFT INC. *(317) 255-6476*
6413 N. College Ave. • Indianapolis, Indiana 46220

VISA

MasterCard

CIRCLE 22 ON READER SERVICE CARD

WALTZ LISP(TM)

The one and only ***adult*** *Lisp system for CP/M users.*

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with *Franz* (the Lisp running under *Unix*), and is similar to *MacLisp*. Waltz is perfect for Artificial Intelligence programming. It is also most suitable for general applications.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix • True dynamic character strings. Full string operations including fast matching/extraction. • Flexibly implemented random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Virtual function definitions. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Assembly language interface. • Over 250 functions in total. • The best documentation ever produced for a micro Lisp (300+ full size pages, hundreds of illustrative examples).

Waltz Lisp requires CP/M 2.2, Z80 and 48K RAM (more recommended). All common 5'' and 8'' disk formats available.

PRO CODE (TM)
INTERNATIONAL

15930 SW Colony Pl.
Portland, OR 97224

Unix* Bell Laboratories.
CP/M* Digital Research Corp.

Version 4.4
(Now includes Tiny Prolog written in Waltz Lisp.)

$169*

*Manual only: $30 (refundable with order). All foreign orders: add $5 for surface mail, $20 for airmail. COD add $3. Apple CP/M and hard sector formats add $15.

Call free **1-800-LIP-4000** Dept. #13
In Oregon and outside USA call 1-503-684-3000

CIRCLE 53 ON READER SERVICE CARD

package) allowing I/O for characters, strings, integers, long integers, and floats

- UCSD strings package (which offers string manipulation routines similar to those found in UCSD Pascal)
- Unit_Io package for low level I/O.

Using the Telesoft Ada compiler is very easy. First, the editor can be used to create and edit a program file, which can be stored in any on-line volume. Afterward, the user exits the editor back to the command shell where the compiler is invoked. Switches and compile options are also available, and the compiler will inform you of any errors with the option to continue compiling.

On occasion when an "unimplemented feature" error appeared, however, I requested to continue compiling. The result was that the system hanged. The compilation for the examples I used was fairly quick, though.

The manual states that the user programs are compiled into p-code instead of native code. I tested the speed of programs running with p-codes. Listing 1 shows the sieve program test, and Listing 2 shows the speeds derived from a program that sorts an array of 1,000 integers. (The array itself is created by the program such that the content of each member equals its array index. This yields a perfectly ascending sorted set. The Ada program's job is to reverse the order, treating the array as if the members had random values.)

Listing 3 shows an example when using floating point math. The program inverts a 50-by-50 matrix with all non-diagonal elements equal to one and the diagonal elements equal to two.

It is important to note that the initial configuration that I set up did not allow the Ada compiler to handle real number calculations. I had to use a script file that upgraded the compiler and established a link between the latter and the 8087 chip. Figure 1 shows the timing results for each program.

I also include for comparison the timing for the same programs as compiled by the Janus/Ada compiler (version 1.4.7). The result shows the advantage gained by using the Janus/Ada compiler, which produces native code. The lead is decreased for the matrix inversion problem since both compilers are using the 8087 chip for floating point math.

Overall, the Telesoft Ada package works fine—but it is expensive. It is not meant for the beginner who wants to learn Ada. However, I am disappointed at the missing features, specifically the lack of dynamic arrays. The compiled p-code is slow, yet the package has a lot of useful utilities and few bugs.

By Namir Clement Shammas

```
WITH TEXT_IO; USE TEXT_IO, INTEGER_IO;

PROCEDURE SIEVE IS
-- Program to perform the sieve test

SIZE : CONSTANT INTEGER := 8190;
TYPE FLAG_TYPE IS ARRAY(0..SIZE) OF BOOLEAN;
I, PRIME, K, COUNT, ITER : INTEGER;
FLAGS : FLAG_TYPE;

BEGIN
    PUT_LINE("START");
    NEW_LINE;
    FOR ITER IN 1..10 LOOP
        COUNT := 0;
        FOR I IN 0..SIZE LOOP
            FLAGS(I) := TRUE;
        END LOOP;
        FOR I IN 0..SIZE LOOP
            IF FLAGS(I) THEN
                PRIME := I + I + 3;
                K := I + PRIME;
                WHILE K <= SIZE LOOP
                    FLAGS(K) := FALSE;
                    K := K + PRIME;
                END LOOP;
                COUNT := COUNT + 1;
            END IF;
        END LOOP; -- End of inner for-loop
    END LOOP;     -- End of outer for-loop
    PUT(COUNT);
    PUT(" Primes");
END SIEVE;
```

Listing 1.

Program	Telesoft Ada	Janus/Ada
Sieve	5'22	0'41
Sort	1'18	0'6
Matrix Inversion	3'59	1'13

Figure 1.

```
WITH TEXT_IO; USE TEXT_IO , INTEGER_IO;

PROCEDURE MYSORT IS
-- Program will test the speed of sorting an integer array.
-- The program will create an array sorted from smaller to larger
-- integers, then sort them in the reverse order.
-- The Shell-Metzner sorting algorithm is used.

MAX : CONSTANT INTEGER := 1000;

TYPE NUMBERS IS ARRAY(1..MAX) OF INTEGER;

DONE : BOOLEAN;
SKIP, I, J, TEMPO : INTEGER;
A : NUMBERS;

BEGIN
    PUT_LINE("Initializing integer array");
    FOR I IN 1..MAX LOOP
        A(I) := I;
    END LOOP;

    SKIP := MAX;
    PUT_LINE("Beginning to sort");
```

Listing 2.

FREE LITERATURE ON MUMPS, ONE OF THE BEST-KEPT SECRETS IN THE COMPUTER INDUSTRY.

UNTIL NOW.

MUMPS is one of an elite circle of languages – like COBOL, FORTRAN and PL/1 – that meets ANSI standards.

Except MUMPS typically outperforms these languages. In speed, lines of code needed, and manipulation of data.

Some 10,000 MUMPS installations exist worldwide. In business, industry and medicine.

Find out how MUMPS can work for you by calling the MUMPS Users' Group or mailing the coupon below for free information packet.

MUMPS Users' Group, Suite 510,
4321 Hartwick Road,
College Park, MD 20740 301-779-6555.

Please send me a free information packet on MUMPS.

NAME ____________________
COMPANY ____________________
STREET ____________________
CITY ____________________ STATE ____________ ZIP ____________

CIRCLE 44 ON READER SERVICE CARD

```
    WHILE SKIP > 1 LOOP
        SKIP := SKIP / 2;
        LOOP -- Open loop equivalent to Repeat-Until in Pascal
            DONE := TRUE;
            FOR J IN 1..(MAX - SKIP) LOOP
                I := J + SKIP;
                IF A(I) > A(J) THEN
                    DONE := FALSE;
                    TEMPO := A(I);
                    A(I) := A(J);
                    A(J) := TEMPO;
                END IF;
            END LOOP;
            IF DONE THEN EXIT; END IF;
        END LOOP; --  End of open loop
    END LOOP; -- End of while-loop

    PUT_LINE("Finished sorting!");
    FOR I IN 1..MAX LOOP
        PUT(A(I));
        PUT(" ");
    END LOOP;

END MYSORT;
```

Listing 2 (Continued).

Discover Forth

Join the **FORTH Interest Group**

The FORTH Interest Group (FIG) is a non-profit member-supported organization, devoted to the Forth computer language. Join our 4700+ members and discover Forth. We provide our members with the information and services they need, including:

Over fifty local FIG chapters (general and special interest) meet throughout the world on a regular basis.

Forth Dimensions magazine is published six times a year and addresses the latest Forth news. A one year subscription to FD is free with FIG membership.

The FIG-Tree is the FIG-sponsored, on-line computer data base that offers members a wealth of Forth information. Dial (415) 538-3580 using a modem and type two carriage returns.

Forth publications: a wide variety of high quality and respected Forth-related publications (listings, conference proceedings, tutorials, etc.) are available.

The FIG HOT LINE (415) 962-8653, is fully staffed to help you.

The Job Registry helps match Forth programmers with potential employers.

All this and more for only $15.00/yr. ($27.00 foreign) Just call the FIG HOT LINE or write and become a FIG member. (VISA or MC accepted.)

Don't miss our upcoming 6th Annual Forth Convention November 16-17, 1984 at the Hyatt Palo Alto in Palo Alto, CA. Call or write for details.

Forth Interest Group

(415) 962-8653
PO Box 1105
San Carlos
CA 94070

CIRCLE 27 ON READER SERVICE CARD

THE MOST EXTENSIVE

THE GREENLEAF FUNCTIONS
Library for C Programmers

Total Access to IBM PC and XT
Compatible with DOS 2.0, 1.1, CI C86, Lattice, and Microsoft C – Versions 1 and 2

PARTIAL CONTENTS

• DOS 2.0 - over 25 functions • Complete Video Access for Text and Graphics •Over 60 String Functions • Rainbow Series Color Text • Time and Date • Over 40 Printer Functions • Function and Special Keys • RS232 Async • All BIOS Functions • Software Diagnostics • Disk functions • Utility functions • and more . . .

THE GREENLEAF FUNCTIONS . . .
Nearly 200 functions, 220 page manual, 3 Libraries, Extensive Examples of each function, Full Source Code

$175.00
Add $7.00 for shipping.
Specify Compiler
MC/VISA Accepted
Prices subject to change without notice.
Dealer Inquiries Welcome.

(214) 446-8641

C LIBRARY ANYWHERE
FOR THE IBM AND PC XT

GREENLEAF SOFTWARE©

GREENLEAF SOFTWARE, INC. • 2101 HICKORY DRIVE • CARROLLTON, TEXAS 75006

CIRCLE 29 ON READER SERVICE CARD

```
WITH TEXT_IO; USE TEXT_IO, INTEGER_IO, FLOAT_IO;

PROCEDURE INVERT IS
-- Program to test speed of floating point matrix inversion.
-- The program will form a matrix with ones' in every member,
-- except the diagonals which will have values of 2

MAX : CONSTANT INTEGER := 50;

TYPE MATRIX IS ARRAY(1..MAX,1..MAX) OF FLOAT;

J, K, L : INTEGER;
DET, PIVOT, TEMPO : FLOAT;
A : MATRIX;

PROCEDURE SHOW_MATRIX IS

BEGIN
    FOR J IN 1..MAX LOOP
        FOR K IN 1..MAX LOOP
            PUT(A(J,K));
            PUT("  ");
        END LOOP;
        NEW_LINE;
    END LOOP;
END SHOW_MATRIX;
```

Listing 3.

• C Programming Guidelines
Thomas Plum

Learning to Program in C
Thomas Plum

C LANGUAGE PROGRAMMING
From Plum Hall...the experts in C training

Learning to Program in C 372 pp., 7½" x 10", Price $25.00
A practical, step-by-step guide for everyone acquainted with computers who wants to master this powerful "implementer's language" Inside, you will learn how to write portable programs for the full spectrum of processors, micro, mini and mainframe

C Programming Guidelines 140 pp., 7½" x 10", Price $25.00
A compilation of standards for consistant style and usage of C language. Arranged in manual page format for easy reference, it presents time-tested rules for program readability and portability.

FREE
C LANGUAGE POCKET GUIDE!
A handy C language programming pocket guide is yours *free* when you order either (or both) of the manuals above. A full 14 pages of valuable C language information!

PLUM HALL
1 Spruce Av, Cardiff NJ 08232
The experts in C and UNIX™ training.
Phone orders: 609-927-3770

Please send me: ______ information on C and UNIX Training Seminars
______ copies of Learning to Program in C @ $25.00/copy
______ copies of C Programming Guidelines @ $25.00/copy
NJ residents add 6% sales tax.

NAME ______
COMPANY ______
ADDRESS ______
CITY/STATE/ZIP ______
☐ Check ☐ American Express ☐ Master Card ☐ Visa
CARD # ______ EXP. DATE ____ / ____ Signature ______
UNIX is a trademark of AT&T Bell Laboratories

CIRCLE 50 ON READER SERVICE CARD

```
   BEGIN

       -- Creating test matrix
       FOR J IN 1..MAX LOOP
           FOR K IN 1..MAX LOOP
               A(J,K) := 1.0;
           END LOOP;
           A(J,J) := 2.0;
       END LOOP;

       -- The test below will ensure that the user does not spend
       -- a lot of time looking at a rather obvious matrix when its
       -- size is large.

       IF MAX <= 10 THEN
           PUT_LINE("Matrix is ");
           SHOW_MATRIX;
           NEW_LINE; NEW_LINE;
       END IF;

       PUT_LINE("Starting matrix invertion");

       DET := 1.0;
       FOR J IN 1..MAX LOOP
           PIVOT := A(J,J);
```

Listing 3 (Continued).

SOURCE SOFTWARE

Are you tired of using inflexible software which you can't modify? Here's the source code for a CP/M-compatible Z-80 assembler of high reliability featuring —

- Standard Zilog mnemonics
- 19 pseudo-op's, including TITLE, XLIST and nested conditionals with ELSE
- Source program can be read from multiple input files
- Prints a sorted symbol table at end of listing
- Modular structure, allowing easy revision as a cross-assembler
- Symbolic definition of all important parameters makes it simple to hand-tailor language features as desired

The complete source listing is contained in a 200-page manual along with a full tutorial explaining top-down how an assembler works. Advanced algorithms such as expression processing by recursive descent are fully explained with illustrations in pseudo-code. The complete source code is also available on a standard format 8" SSSD diskette.

Z-80 Assembler Manual	$25
Manual and Diskette	$50

(Foreign orders add $3 for surface mail or $10 for airmail)

King Software
PO Box 208
Red Bank, N.J. 07701
(201) 530-7245

NJ residents please add 6% sales tax

CIRCLE 33 ON READER SERVICE CARD

LATTICE®

C Compilers

"My personal preferences are Lattice C in the top category for its quick compile and execution times, small incremental code, best documentation and consistent reliability;..."

BYTE AUG. 1983
R. Phraner

"... programs are compiled faster by the Lattice C compiler, and it produces programs that run faster than any other C compiler available for PC-DOS."

PC MAGAZINE JULY 1983
H. Hinsch

"... Microsoft chose Lattice C both because of the quality of code generated and because Lattice C was designed to work with Microsoft's LINK program."

PC MAGAZINE OCT. 1983
D. Clapp

"Lattice is both the most comprehensive and the best documented of the compilers. In general it performed best in the benchmark tests."

PERSONAL COMPUTER AGE NOV 1983
F. Wilson

"This C compiler produces good tight-running programs and provides a sound practical alternative to Pascal."

SOFTALK AUG 1983
P. Norton

"... the Lattice compiler is a sophisticated, high-performance package that appears to be well-suited for development of major application programs."

BYTE AUG 1983
Houston, Brodrick, Kent

To order, or for further information on the LATTICE family of compilers, call or write:

LATTICE, INC.
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950 TWX 910-291-2190

CIRCLE 36 ON READER SERVICE CARD

```
        DET := DET * PIVOT;
        A(J,J) := 1.0;
        FOR K IN 1..MAX LOOP
          A(J,K) := A(J,K) / PIVOT;
        END LOOP;
        FOR K IN 1..MAX LOOP
          IF K /= J THEN
            TEMPO := A(K,J);
            A(K,J) := 0.0;
            FOR L IN 1..MAX LOOP
              A(K,L) := A(K,L) - A(J,L) * TEMPO;
            END LOOP;
          END IF;
        END LOOP; -- End of inner for-loop
      END LOOP; -- End of outer for-loop

      PUT_LINE("The inverse matrix is ");
      SHOW_MATRIX;
      PUT("Determinant = ");
      PUT(DET);
      NEW_LINE; NEW_LINE;
    END INVERT;
```

Listing 3 (Continued).

Scroll & Recall™

Screen and Keyboard Enhancement for the IBM - PC, XT and Compatibles

Allows you to conveniently scroll back through data that has gone off the top of your display screen.

Allows you to easily recall and edit your previously entered DOS commands and data lines.

Very easy to use, fully documented. Compatible with all versions of DOS, monochrome & graphic displays.

***S69** - Visa, M/C, Check, COD, POs*
Phone orders accepted

Make Your Work Easier!

To Order or to Receive Additional Information, Write or Call:

Opt-Tech Data Processing
P.O. Box 2167 • Humble, Texas 77347
(713) 454-7428
Dealer Inquiries Welcome

CIRCLE 47 ON READER SERVICE CARD

SUPER FORTH 64®

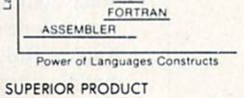
TOTAL CONTROL OVER YOUR COMMODORE-64™
USING ONLY *WORDS*
MAKING PROGRAMMING FAST, FUN AND EASY!

MORE THAN JUST A LANGUAGE...
A complete, fully-integrated program development system.
Home Use, Fast Games, Graphics, Data Acquisition, Business
Real Time Process Control, Communications, Robotics, Scientific, Artificial Intelligence

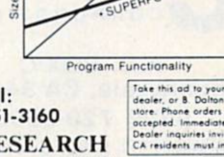
A Powerful Superset of MVPFORTH/FORTH 79 + Ext. for the beginner or professional

- 20 to 600 x faster than Basic
- 1/4 x the programming time
- Easy full control of all sound, hi res. graphics, color, sprite, plotting line & circle
- Controllable SPLIT-SCREEN Display
- Includes interactive interpreter & compiler
- Forth virtual memory
- Full cursor Screen Editor
- Provision for application program distribution without licensing
- FORTH equivalent Kernal Routines
- Conditional Macro Assembler
- Meets all Forth 79 standards+
- Source screens provided
- Compatible with the book "Starting Forth" by Leo Brodie
- Access to all I/O ports RS232, IEEE, including memory & interrupts
- ROMABLE code generator
- MUSIC-EDITOR
- SPRITE-EDITOR
- Access all C-64 peripherals including 4040 drive
- Single disk drive backup utility
- Disk & Cassette based. Disk included
- Full disk usage — 680 Sectors
- Supports all Commodore file types and Forth Virtual disk
- Access to 20K RAM underneath ROM areas
- Vectored kernal words
- TRACE facility
- DECOMPILER facility
- Full String Handling
- ASCII error messages
- FLOATING POINT MATH SIN/COS & SQRT
- Conversational user defined Commands
- Tutorial examples provided, in extensive manual
- INTERRUPT routines provide easy control of hardware timers, alarms and devices
- USER Support

SUPER FORTH 64® *is more powerful than most other computer languages!*



SUPER FORTH 64® *compiled code becomes more compact than even assembly code!*



A SUPERIOR PRODUCT in every way! At a low price of only
$96

Call:
(415) 651-3160

PARSEC RESEARCH
Drawer 1776, Fremont, CA 94538

Take this ad to your local dealer, or B. Dalton Bookstore. Phone orders also accepted. Immediate delivery! Dealer inquiries invited. CA residents must include tax.

or C.O.D.

® PARSEC RESEARCH (Established 1976)

Commodore 64 & VIC-20 TM of Commodore

CIRCLE 49 ON READER SERVICE CARD

DeSmet C

8086/8088 Development Package $109

FULL DEVELOPMENT PACKAGE
- Full **K&R** C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete **STDIO** Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 AND SOFTWARE FLOATING POINT

OUTSTANDING PERFORMANCE
- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER $50
- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT $35
- Uses DOS .OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

Check: ☐ Dev. Pkg (109)
☐ Debugger (50)
☐ DOS Link Supt. (35)

SHIP TO: ____________

____

____________ ZIP ________

C WARE CORPORATION

P.O. BOX C
Sunnyvale, CA 94087
(408) 720-9696

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5, elsewhere add $15. Checks must be on US Bank and in US Dollars. Call 9 a.m. – 1 p.m. to CHARGE by VISA/MC/AMEX.

CIRCLE 18 ON READER SERVICE CARD

Volition System's Modula-2

Hardware required: IBM PC or PC/XT with 2 double-sided drives, 128K, and PC-DOS 2.0 or 2.1; Apple II, II+, IIe, IIc, III (all require Apple Pascal); Sage II and IV
Price: $395 (IBM), $295 (Apple), $495 (Sage)
Available from: Volition Systems, P.O. Box 1236, Del Mar, Calif. 92014. (619) 481-2286
Support: No software warranty, 30-day media warranty from purchase date, updates available for $25-$50 depending on whether documentation is necessary

Modula-2 by Volition Systems is considerably more than just a Modula-2 compiler. It is a complete software development system for Modula-2 built on the UCSD Pascal operating system. This system is based on an imaginary machine that is an optimal environment for UCSD Pascal.

Volition's Modula-2 comes with ASE, the Advanced Systems Editor, which is a screen-oriented text editor that edits files that are as large as the available disk space, in contrast with the standard p-system editor. ASE has many powerful features and supports user-defined macros.

Volition's Modula-2 is a full implementation of the standard Modula-2 as defined by N. Wirth and supports the following features: interrupts; one-pass compiler; erroneous source code displayed with syntax error messages; conditional compilation, no linking necessary; source code compatability between Apple, IBM, and Sage; comprehensive module library and library manager utility; full set of program development utilities; and complete documentation including Wirth's *Programming in Modula-2*.

Since this Modula-2 runs under the p-system, the compiler produces an object program coded to run on the pseudo-machine called the p-machine. Volition's system is constructed with an emulator program for the p-machine, which interprets the p-machine object code into specific machine code instructions for the particular machine environment.

Because the interpretation process requires processor time, its programs have a slower execution time compared with native-code Modula-2 compilers. An example of this is provided by the Sieve of Eratosthenes benchmark execution timings on the Volition System compiler (185 sec). For comparison,the time on Logitech's native-code Modula-2 compiler is 16 sec.

Naturally, there is a price to pay for increased execution speed by using a native-code compiler: increased effort to compile programs. Logitech has a four-pass compiler, and you must explicitly link to your program libraries. Other considerations must be taken into account also such as the relative ease of source code syntax error correction and the general nature of the development system environment.

Volition's Modula-2 requires 64K or more of RAM memory, 2 double-sided disk drives for the IBM PC, and 128K and PC-DOS 2.0 or 2.1 for the PC XT. Compiled Modula-2 programs can be transported in most cases to or from both Apple and Sage systems from the IBM PC. Modula-2 for the IBM PC and XT runs as a subsystem under PC-DOS 2.0 and allows programs to transparently access DOS data files. Modula-2 programs can be configured to operate as DOS applications.

A separate Modula-2 package is available for IBM PCs with single-sided disk drives. This system runs as a stand-alone operating system and does not provide DOS access.

Data space is limited to 64K bytes and code space is limited only by memory size. RAM disk support is included along with double precision floating point arithmetic via the 8087 numeric co-processor.

This review was done using the IBM PC version of Modula-2, Release 0.3, on a two-disk-drive IBM PC with 256K RAM. Installation for the PC is done by making back-ups of the distribution diskettes and providing work disks for compilation and intermediate files. After preparing work copies, you must rearrange the system disk files to provide for efficient disk storage space.

Sections in the manual give instructions on how to reconfigure your Modula-2 system for faster disk I/O, RAM disk, larger memory spaces, or different peripherals. A caution is given that this reconfiguration should only be attempted by those with a fair amount of programming knowledge.

Sample programs are also supplied in source code form on the distribution diskettes to get started compiling quickly.

The first step in compiling is to compose your source code with the text editor ASE. The Modula-2 system is invoked from the operating system prompt (>) by typing:

```
> m2 vs \ system
```

As a DOS subsystem, Modula-2 stores its files on virtual disk volumes (i.e., DOS data files that contain a file directory and a number of Modula-2 files). After startup, the system displays the names of all on-line disk volumes and the current system date. The system prompt line then appears across the top of the screen:

```
Xecute, Batch, Shell, Run, File, Edit,
     Comp, UsrRst, Init
```

The following explains the prompt line:

- (*X*)ecute executes the specified code file. The following prompt appears:

```
Execute what file?
```

- (*B*)atch invokes the batch command interpreter. The batch command interpreter is the code file SYSTEM.BATCH on the system volume
- (*S*)hell invokes the shell command interpreter, which is the code file SYSTEM.SHELL on the system volume
- (*R*)un executes the work file, which is a special file used as a "work" area for developing programs
- (*F*)ile invokes the filer, which manages disk files and volumes
- (*E*)dit invokes the editor, which is the ASE text editor
- (*C*)omp invokes the Modula-2 compiler
- *UrRst* re-executes the last program executed
- (*I*)nit causes the system to reinitialize its state information.

The Linker utility is used to link together an interpreter skeleton and drivers into a complete interpreter file. Linking is not normally required for Volition's Modula-2 programs because module binding is performed at run-time.

The program is executed from the system prompt line and is interpreted by the pre-configured interpreter, which can be tailored to your own desires.

Many other functions of Modula-2 are available to handle process control as well as other programming tasks that require concurrency. Modula-2 also includes features to set the compiler options, reconfigure the operating system interface, and implement a large number of DOS standard interrupts. This allows real-time programming in Modula-2, while the system-dependent library modules provide access to the UCSD Pascal file system and the UCSD Pascal intrinsics.

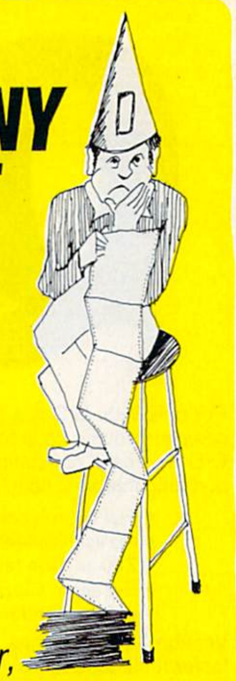
The standard library modules that provide Modula-2 with a standard operating environment are implemented as interfaces to an underlying operating system. Because all implementations present the same module interfaces, programs that use the standard library modules are portable across all Modula-2 systems. The standard module library allows the following functions: console I/O, file I/O, storage management, code management, exception handling, process scheduling, strings, decimal arithmetic, math functions, and dynamic module linking.

The Modula-2 one-pass compiler flags syntax errors with a descriptive error message and displays the program text where the error was found. The compilation can either be continued or the program text can be immediately edited. The system also displays an explanatory error message for execution errors, giving the current module and procedure name, the code offset of the error and a procedure call chain. Run-time checks are provided for value range errors, floating point errors, string overflows, and stack/heap overflow.

In summary, the Modula-2 system by Volition is a fast program development environment for the Modula-2 or Pascal programmer. A number of features are provided for the software application developer to handle many types of jobs. The speed of the compilation process coupled with the quick and easy error correction process makes for rapid development.

The ASE full screen program editor provides powerful text editing capabilities formerly only available on larger computers. This complete programming package includes an operating system, utilities complete with documentation, as well as the ASE manual. The documentation is extensive and comprehensive. Volition has created a superior development environment.

By Chris Jacobs



WHY JOHNNY CAN'T READ HIS OWN CODE

Johnny's A Good Programmer, Even Brilliant, But—Johnny works in 8080/Z80 assembly language, with a conventional assembler. That can make yesterday's brilliance today's garble, a maze of mnemonics and a jumble of meaningless labels. Johnny's program is less than self-explanatory—even for Johnny.

Johnny *could* read his own code if he used SMAL/80—the *superassembler*—and so can you. SMAL/80 boosts your program's clarity and your productivity by giving you:

- Familiar algebraic notation in place of cryptic mnemonics—"A=A−3" for example, instead of "SUI 3" (if you know BASIC or Pascal, you *already* know SMAL/80)
- Control structures like BEGIN...END, LOOP...REPEAT WHILE, and IF...THEN...ELSE... to replace tangled branches and arbitrary label names (eliminating up to 90% of labels with *no* overhead imposed)
- Complete control over your processor—because SMAL/80 is a true assembler, it doesn't reduce execution speed or burden your program with its own runtime routines.

SMAL/80, the assembler that handles like a high-level language, lets you do it right the first time, and lets you read and understand your work afterward—the next day or a year later. Users say SMAL/80 has doubled and even tripled their output of quality code. But don't take our word for it—***TRY IT!***

Use SMAL/80 for 30 days. If you're not *completely* satisfied with it—for *any* reason—return the package for a full refund.

SPECIAL BONUS: Order before Dec. 31, 1984, and get *Structured Microprocessor Programming*—a $25 book ***FREE!***

SMAL/80 for CP/M-80 systems (*all* CP/M disk formats available—please specify); produces 8080/8085 and Z80 code. Now supports Microsoft .REL.**ONLY $149.95**

SMAL/80 for CP/M-80 systems, 8080/8085 output only. **SAVE $20: $129.95**

NEW! SMAL/80X65—for Apple II and IIe (requires Z80 card and CP/M); produces Z80 *and* 6502 object code. **$169.95**

Mastercard
Visa
C.O.D.'s

SMAL/80
CHROMOD ASSOCIATES
(201) 653-7615
1030 Park Ave. Hoboken, N.J. 07030

We pay shipping on prepaid orders

CIRCLE 10 ON READER SERVICE CARD

Q/C

For only $95, Q/C is a ready-to-use C compiler for CP/M. You get complete source code for the compiler and over 75 library functions. Q/C is upward compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, and bit fields.

- Full source code for compiler and library.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Excellent support for assembly language and ROMs.
- Q/C is standard. Good portability to UNIX.

Version 3.2 of Q/C has many new features: structure initialization, faster runtime routines, faster compilation, and improved ROM support. Yes, Q/C has casts, typedef, sizeof, and function typing. The *Q/C User's Manual* is available for $20 (applies toward purchase). VISA and MasterCard welcome.

THE CODE WORKS

5266 Hollister
Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q C, CP M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

CIRCLE 11 ON READER SERVICE CARD

COMPUTER RESOURCES *of WAIMEA*

* * * EASY TO USE * * *
Macro Programs for
Spellbinder™

We have been using and working with Spellbinder since late 1981. We use computers extensively in the day-to-day operation of our business and have developed a number of programs which we find useful. We recently formed a software development and marketing company - Computer Resources of Waimea, to promote and market these programs, most being enhancements and macro programs running under Spell binder. Spellbinder's macro programming language M-Speak is extremely versatile and in our opinion is one of the best kept "secrets" in the world of micro computers. We have a number of macro programs for the end user, a number of utilities for the programmer, and for those who want a more or less organized instruction set for M-Speak, our head programmer has compiled his personal notes into a booklet which the M-Speak user should find very useful. It can be purchased for $10.00. Send for our complete listing.

P.O. Box 1206 Kamuela, Hawaii 96743
(808) 885-7905

CIRCLE 14 ON READER SERVICE CARD

UNIX SOFTWARE

UniPress Product UPDATE

LATTICE® C NATIVE AND CROSS COMPILERS FOR THE 8086
AMSTERDAM COMPILER KIT

Outstanding software development tools

Lattice C Cross Compiler to the IBM-PC

- ■ Highly regarded compiler producing fastest and tightest code for the 8086 family.
- ■ Use your VAX or other UNIX machine to create standard Intel object code for your 8086 (IBM-PC)
- ■ Full C language and standard library, compatible with Unix.
- ■ Small, medium, compact and large address models available.
- ■ Includes compiler, linker, librarian and disassembler.
- ■ 8087 floating point support.
- ■ MS-DOS 2.0 libraries included.
- ■ Send and Receive communication package optionally available.

Prices:	Hosted On	
	VAX/Unix and VMS	$5000
	MC68000/8086	3000
	Send and Receive	500

Lattice C Native Compiler for the 8086

- ■ Runs on the IBM-PC under MS-DOS 1.0 or 2.0.
- ■ Produces highly optimized code
- ■ Small, medium, compact and large address models available.
- ■ Compiler is running on thousands of 8086 systems.

Price: $425

Plink (Optional)

- ■ Full function linkage editor including overlay support.

Price: $395

Amsterdam Compiler Kit

- ■ Package of compilers, cross compilers and assemblers.
- ■ Full C and pascal language.
- ■ Generates code for VAX, PDP-11, MC68000, 8086 and NSC16000.
- ■ Hosted on many Unix machines.
- ■ Extensive optimization.

Price:		
	Full system	$9950
	Educational Institution	995

OEM terms available • Much more Unix software, too! • Call or write for more information.

Mastercard and Visa

UniPress Software, Inc.

2025 Lincoln Highway, Edison, NJ 08817
201-985-8000 • Order Desk: 800-222-0550 (outside NJ) • Telex 709418

Lattice is a registered trademark of Lattice, Inc. Unix is a trademark of Bell Laboratories. MS-DOS is a trademark of Microsoft.

UNIX SOFTWARE

CIRCLE 66 ON READER SERVICE CARD

SNOBOL4+

Hardware required: IBM PC or other 8086-, 8088-, or 80186-based computers. A minimum of 128K and at least one single-sided, double-density 5¼-in. disk drive are also required

Price: $95, $3 additional for shipping

Available from: Catspaw Inc., P. O. Box 1123, Salida, Colo. 81201, (303) 539-3884

This is a review of a new product that's been around well over two decades!

Contradiction? Not really. Mark Emmer of Catspaw Inc. has just produced a new and excellent 8086/88/186 implementation of a language first introduced in the early 1960s. Better yet, its price is better than reasonable: $50.

If the programs you're writing are loaded with string manipulation, general text processing, and pattern recognition, then read on.

Although primarily a review of a specific implementation, this review will very briefly examine the SNOBOL language, as defined by one of its originators, Dr. Ralph E. Griswold (also see "Discovering SNOBOL4" in the premier issue of *COMPUTER LANGUAGE*, pp. 65-68). With this base definition, I'll then try to "score" the Catspaw implementation, giving my subjective pluses and minuses for this specific product.

IBM PC compatibility is actually not required; SNOBOL4+ will operate on machines such as the TI Professional and the Tandy 2000. (PC DOS or Microsoft DOS Versions 1.10, 1.25, 2.00, or 2.10 are supported.)

Catspaw also suggests that at least 192K is needed for programs of a "reasonable size," and says that the compiler will utilize available memory up to a maximum of approximately 410K bytes. I'm running the system on an IBM PC (with PC DOS V2.0) with three double-surface, double-density drives and 640K memory. The primary file (the compiler itself) on the distribution disk is about 73K in size. So unless you enjoy disk swapping, a two-drive system appears extremely desirable.

An interesting, and to some of you, critical point: the 8087 co-processor is fully supported but not required!

Finally, the latest distribution package includes a SNOBOL4 library of the programs contained in Dr. Griswold's *String and List Processing in SNOBOL4*. This provides many excellent examples of the power of the language and the subtleties of implementation.

So now that you know you can run the product, what will it do for you? The scope of this review cannot possibly cover the full language. There are several excellent books available that give full details. I've included two at the end of this article. In this review I'll cover a few of the unusual features; if these fit your needs, get one or more of the reference books to make your final decision on the language.

So what will it do? Here are some examples:

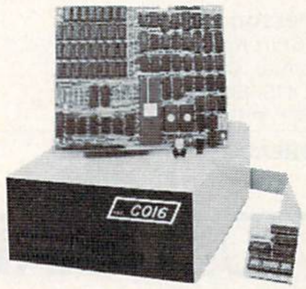
Want to find literally any defined pattern in a text string? SNOBOL will do it, in one simple function. Want to translate from EBCDIC to ASCII? SNOBOL will do it, in one simple function. Want to manipulate, concatenate, extract, replace, substitute, etc., any combination of strings? SNOBOL will do it, typically in one simple "built in" function.

The key to all of these things is the phrase "one simple function." SNOBOL, for this type of process, is one of the most concise languages I've encountered. A program to copy a straight ASCII text file might readily be a single statement.

Although string manipulation is always emphasized as the major strength of SNOBOL, this emphasis effectively hides many other very significant features. SNOBOL may well be unique in its ability to accept strings that are variable names

A POWERFUL 68000 DEVELOPMENT ENVIRONMENT FOR YOUR Z80 SYSTEM

CO1668 ATTACHED RESOURCE PROCESSOR



- 68000 Assembler
- C Compiler
- Forth
- Fortran 77

- Pascal
- BASIC-PLUS
- CBASIC
- APL. 68000

6 MHZ 68000 CP/M-68K 768K RAM
4 x 16081 MATH CO-PROCESSORS CPM80 RAM DISK

Develop exciting 68000 applications on your current Z80 based CPM system using powerful mini-frame like 32 bit programming languages. And then, execute them at speeds that will shame many $100K plus *mini*computer systems.

The CO1668 ATTACHED RESOURCE PROCESSOR offers a Z80 CPM system owner a very low cost and logical approach to 68000 development. You have already spent a small fortune on 8 bit diskette drives, terminals, printers, cards cages, power supplies, software, etc. The CO1668 will allow you to enjoy the vastly more powerful 68000 processing environment, while preserving that investment.

CO1668 ATTACHED RESOURCE PROCESSOR SPECIAL FEATURES:

- 68000 running at 6 Mhz
- 256K to 768K RAM (user partitioned between CPU and RAM Disk usage)
- Up to four 16081 math co-processors
- Real time clock, 8 level interrupt controller & proprietory I/O bus
- Available in tabletop cabinet
- Delivered w/ sources , logics, & monolithic program development software
- Easily installed on *ANY* Z80 CPM system
- CP/M-68K and DRI's *new* UNIX V7 compatible C complier (w/ floating point math) - standard feature
- *Can be used as 768K CPM80 RAM Disk*
- Optional Memory parity
- No programming or hardware design required for installation
- Optional 12 month warrantee

- PRICES START AS LOW AS $899.00 for a CO1668 with 256K RAM, CPM68K, C Complier, Sources, Prints, 200 page User Manual, Z80 Interface, and 68000 System Development Software.

For further information about this revolutionary product or our Intel 8086 Co-Processor, please send $1 [no checks please] or call:

HSC INC.

Hallock Systems Company, Inc.
262 East Main Street
Frankfort, New York 13340
(315) 895-7426

RESELLER AND OEM INQUIRIES INVITED.

CIRCLE 31 ON READER SERVICE CARD

QUALITY SOFTWARE AT REASONABLE PRICES

CP/M Software by

Poor Person Software

Poor Person's Spooler All the function of a hardware print buffer at a fraction of the cost. Keyboard control. Spools and prints simultaneously.	**$49.95**
Poor Person's Spread Sheet Flexible screen formats and BASIC-like language. Preprogrammed applications include Real Estate Evaluation.	**$29.95**
Poor Person's Spelling Checker Simple and fast! 33,000 word dictionary. Checks any CP/M text file.	**$29.95**
aMAZEing Game Arcade action for CP/M! Evade goblins and collect treasure.	**$29.95**
Crossword Game Teach spelling and build vocabulary. Fun and challenging.	**$39.95**
Mailing Label Printer Select and print labels in many formats.	**$29.95**
Window System Application control of independent virtual screens.	**$29.95**

All products require 56k CP/M 2.2 and are available on 8" IBM and 5" Northstar formats, other 5" formats add $5 handling charge. California residents include sales tax.

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

CIRCLE 51 ON READER SERVICE CARD

→ Syntax → *custom software*
Constructs ←

Get the power of your Z80 and the elegance of direct access to CP/M functions from your high level programs with

SYNLIB
utility library

***SYNLIB** consists of MICROSOFT compatible object code that may be called from any high level language that uses MICROSOFT parameter passing conventions.*

***SYNLIB** gives you extremely powerful array and buffer manipulation using the Z80 LDIR instruction; program access to the CP/M CCP console command line; high speed disk block I/O; a high quality random number generator; HEX to ASCII conversion optimized by special Z80 instructions; program chaining and more.*

And, because our programmer abhors a vacuum, each 8" floppy comes packed with MODEM7 and other valuable public domain software. We've included documentation and available source, so that you too may join the free software movement.

SYNLIB *$50.00*
8" SSSD CP/M format
SOURCE: $100.00

Licensing for commercial use available.

SYNTAX CONSTRUCTS, INC.
14522 Hiram Clarke
Houston, Texas 77045
(713) 434-2098

CP/M is a registered trademark of Digital Research, Inc. Microsoft is a registered trademark of Microsoft Corp. Z80 is a registered trademark of Zilog.

CIRCLE 63 ON READER SERVICE CARD

Use ALL the Power of Your MS-DOS, IBM PC-DOS, or CP/M-80 System with UNIX-Style Carousel Tools

CAROUSEL TOOL KIT
CAROUSEL TOOL KIT
SOFTWARE TOOLS FOR MICROS
CAROUSEL TOOL BOOK

```
ch "CP/M" "MS-DOS" <doc >newdoc
diff newdoc doc I more
ed newdoc
kwic newdoc I sortmrg I uniq I unrot >index
make -f makdoc ndx
```

CAROUSEL TOOLS are a proven set of over 50 programs designed to be used with pipes, redirected I/O and scripts. In the style of UNIX each Tool does one thing well, and the Tools can be used together to do more complex tasks.

YOU ACCOMPLISH MORE using Carousel Tools: better programming and documentation support, simpler data and file housekeeping, more general file handling.

TOOLS FOR PC/MS-DOS 2.x AND CP/M-80 are available now. The DOS ToolKit is $149. The CP/M ToolKit is $249 and includes a *shell* to provide pipes, redirected I/O, and scripts. Source code is available for $100 more.

ORDER YOUR TOOLKIT TODAY. VISA

CALL OR WRITE:

CAROUSEL MICROTOOLS, INC.
609 Kearney Street, El Cerrito, CA 94530 (415) 528-1300

Carousel Tools and Carousel ToolKits are trademarks of Carousel MicroTools, Inc. CP/M is a trademark of Digital Research; IBM is a trademark of International Business Machines; MS is a trademark of MicroSoft; UNIX is a trademark of Bell Laboratories.

CIRCLE 8 ON READER SERVICE CARD

or statement labels even in its input stream and to use these strings in their syntactical sense in program execution. This is the "computed *GOTO*" with a vengeance!

How about indirect addressing? Quite a number of years ago, hardware vendors included this capability in the hardware, but few compilers took advantage of it. SNOBOL implements this ability as a natural part of its syntax and extends it indefinitely in depth. For example, one can assign data to a variable. But rather than actually assigning to the variable, one may choose to assign it to the variable contained in the variable, contained in the variable . . . for an indefinite depth of iteration.

One more: have you ever had the need to select common elements from a set? For example, what is the frequency of word usage in this article? Care to write a program to do the count? I've written this one several times. It's a useful tool to determine overuse of pet phrases.

Typically, one is forced into a alpha/beta tree or some other such structure, with a fair degree of complexity implied. Well, hold on to your hat . . . SNOBOL does it almost "automatically". SNOBOL has a structure called TABLE. It is probably most easily thought of as a single dimensioned array, but with one major difference. The subscript used with the array is the data itself! You don't even have to know how many unique entries you'll have.

By at least one definition, this is associative memory. Are you artifical intelligence types listening? The compiler even includes a direct and simple method of converting the resultant table into a true multi-dimensioned array so more conventional processing methods can be applied to the end result.

One architectural limitation of the language should be considered. As background, it should be stated that the compiler we're discussing is actually not a true compiler in the sense of a product that produces directly executable machine code. Rather, this compiler produces a dense, internal form of the program and then executes this form by interpretation.

Conceptually similar to the Pascal compiler and the p-machine structure, this typically is a reasonable trade-off between speed of execution, ease of modification, portability, etc. It is, however, an "interpretive" execution and therefore slower than a directly executable machine code implementation.

How slow is slow? The standard answer is, "depends on the instruction (function) mix," and that is certainly true here. Subjectively, the implementation produces results considerably faster than interpretive BASIC and seems roughly equivalent of the speeds realized from compiled BASIC.

To be more specific, the attached program processed a text file of slightly less than 16,000 characters in 222 sec. This included the time required to list the resultant word list on the display (60 sec). The function was clearly "process bound" since the disk was active only during a small portion of the total execution time.

Because of the unique nature of the language, finding a comparison benchmark that really makes a valid comparison is difficult. It would be useless and misleading to make a comparison against a "number crunching" application written in FORTRAN, for example. Similarly, to attempt to use FORTRAN to do the primitive text scan of the included program would be totally unfair to FORTRAN.

I have written a C language program that essentially performs the same function as the included SNOBOL program. This implementation uses the CI C compiler and produces native object code for direct execution. This program, run against the same file, produces its results in 93 sec.

This seems to be a criticism of SNOBOL. But to understand the whole issue, one must also know that the C program is composed of five relatively complex and considerably larger procedures whose listing is about six pages of printer output. When compared to the primitive and simplistic program included with this review, it may be that you will consider this factor a major strength.

Now that you've been impressed by the power, what about the weaknesses? Well, the language does not possess even rudimentary structural forms. *IF-THEN-ELSE* constructs, for example, are not supported.

The syntax is simplistic, and you may readily create completely readable code. This and several other "rules" of the language are carryovers from the very early implementations. At worst, a few are annoying, but none are truly severe weaknesses.

Now let's look at Catspaw's implementation of this language. Emmer, the product's original designer, says:

> "[This compiler] has all the features of mainframe SNOBOL4, plus numerous additional features. Compatibility with mainframe SNOBOL4 is achieved by basing this product on the Macro Implementation used on such mainframes as the IBM 360 and CDC 6600. Thus, SNOBOL4+ incorporates a thoroughly tested implementation in its entirety. This also ensures full compatibility with subsequent mainframe SNOBOL4 products and releases."

From the somewhat limited and cursory use I was able to make of the product, I would accept this statement as accurate and fair.

All too often the personal computer

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

Train Your Computer to be an EXPERT!

Expert systems facilitate the reduction of human expertise to simple, English-style rule-sets, then use them to diagnose problems. "Knowledge engineers" are developing many applications now.

EXPERT-2, Jack Park's outstanding introduction to expert systems, has been modified by MMS for MMSFORTH V2.0 and up. We supply it with full and well-documented source code to permit addition of advanced features, a good manual and sample rule-sets: stock market analysis, a digital fault analyzer, and the Animal Game. Plus the benefits of MMSFORTH's excellent full-screen editor, super-fast compiling, compact and high-speed run-time code, many built-in utilities and wide choice of other application programs.

(Rule 1 - demo in EXPERT-2)
IF YOU WANT EXPERT-2
ANDNOT YOU OWN MMSFORTH
THENHYP YOU NEED TO BUY
MMSFORTH PLUS EXPERT-2
BECAUSE MMSFORTH IS REQUIRED

EXPERT-2 in mmsFORTH

Another exciting tool for our alternative software environment!

- Personal License (required):
  MMSFORTH System Disk IBM PC $249.95
  MMSFORTH System Disk TRS-80 1, 3 or 4 . . $129.95
- Personal License (optional modules):
  FORTHCOM communications module $39.95
  UTILITIES $39.95
  GAMES $39.95
  EXPERT-2 expert system $69.95
  DATAHANDLER $59.95
  DATAHANDLER-PLUS (for PC only) $99.95
  FORTHWRITE word processor $175.00
- Corporate Site License Extensions . . . . from $1,000

Shipping/handling & tax extra.

Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

PROGRAMMER'S DEVELOPMENT TOOLS

IBM Personal Computer Language and Utility Specialists

LANGUAGES:

	List	Ours
C-86 Computer Innovations	$395	319
C Programming System by Mark Williams	500	459
Lattice C Compiler	500	299
Professional BASIC Morgan Computing	345	295
ADA-86 + Tools Janus	700	499

Call for Microsoft and Digital Research Products.

*** "C" Language Starter Kit ***

Package Consists of:	List	Ours
DeSmet C Compiler w/Debugger	$159	145
Windows For C Creative Solutions	150	119
C Programming Language book by K & R	25	20

Retail $334, Priced Separately $284

Our Special Package Price! $269

Greenleaf Utilities available for DeSmet C. Call for Details and Prices.

We have in-staff APL expertise!

**** STSC APL*Plus/PC ****

This powerful, interactive, fourth-generation language includes a tutorial, help system and useful extensions. Comes with plug-in APL character generator chip.

Retail $595 Our Normal Price $540

Special Sale Price! $469

Send for complete demonstration package $5

UTILITIES:

C Functions Library by Greenleaf	175	159
Btrieve by SoftCraft	245	205
Communications Library by Greenleaf *New*	160	139
Trace-86 by Morgan Computing	125	115
OPT-TECH Sort *High Performance Utility*	99	87
Profiler by DWB & Associates	175	149
AKA ALIAS by Soft Shell Technology	60	57
Plink-86 *Overlay Linkage Editor*	395	315
Panel *Screen Design/Editing* by Roundhill	350	259
FirsTime *Intelligent C Text Editor* by Spruce	295	CALL
Halo *Color Graphics for Lattice, CI-86*	200	159
Windows For C by Creative Solutions	150	119

*** A SOLID GOLD VALUE ***

CodeSmith-86 Debugger
Version 1.8 by Visual Age

Retail $145, Our Normal Price $129

Special Sale Price! $109

Prices are subject to change without notice.

Account is charged when order is shipped.

Visa/MC
NO EXTRA CHARGE

CALL FOR LOW PRICES
1-800-336-1166

Programmer's Connection
281 Martinel Drive
Kent, Ohio 44240
(216) 678-4301 (In Ohio)

"Programmers Serving Programmers"

CIRCLE 54 ON READER SERVICE CARD

versions of languages omit the more difficult or costly features of the prototype compilers. This is certainly not true in this case. Catspaw has even added needed functions aimed specifically at the personal computer environment. The power of the SNOBOL language plus this excellent (and large—73K) implementation are clearly major strengths of the product.

To put the compiler in fair context, however, a few of the representative limits imposed by the implementation are listed in Table 1. These limits are generally large enough to be ignored in most applications.

The distribution disk also includes several aids for the user. This compiler can even permit the introduction of assembler programs into its environment.

Compilation speed is fast—so fast, in fact, that for the relatively trivial program I've used to benchmark this product, I'm forced to rely on the compiler statistics for time. The program included with this article compiles in less than 3 sec after the compiler is actually loaded and in control of the machine.

The *DUMP* and *TRACE* functions of the compiler are useful, with particular emphasis on the *TRACE* function for execution time debugging. The program I've included is sufficiently simple that my use of *TRACE* was purely experimental, but it appears conceptually that this will be a very useful tool.

Emmer stresses that his manual is not a tutorial. He has done an excellent job of writing a concise and definitive document, and he is also very correct. If you're attempting to learn the language, this manual is not the place to start!

Emmer is currently working on an introductory section for his manual, along with a demonstration program. This should help, but I'm sure you'll still need the reference manuals to really use the product.

The program in Listing 1 is a trivial and crude example of SNOBOL. This example is included purely to illustrate the power of SNOBOL. Line 1 tells the compiler to "trim" off trailing blanks from input records. Lines 2 and 3 establish a pattern of all of the valid alphabetic characters, while line 4 defines the use of this pattern for extracting "words" from the text.

In English, this definition simply defines a word as being an alphabetic character string bounded by non-alpha characters. Line 5 defines a TABLE, COUNT which is to have 100 entries initially and is to grow 100 entries at a time.

Line 6 defines an external data file with a record length of 80. The next three lines (8-11) are really the total program. Line 8 reads a record. Line 9 replaces all upper case characters with lower. Line 10 extracts the next "word", and Line 11 increments the count for that word in the table.

Note that we have no idea what the word is or how many different words we'll encounter! The program flow is such that we read a record and then *NEXTW* extracts the next word from the record. This word is placed in the table by line 11, and control is returned to *NEXTW*. The program loops between line 10 and 11 until the word extraction "fails". Control then takes the fail exit *F(READ)* and returns to the *READ* statement to read the next record. The process continues until the *READ* statement fails (end of file), and then control is transferred to *PRINT*.

The statement labeled *PRINT* converts the TABLE to a normal two dimensional array, and then the *PRTC* statement displays the word and its count. Note the relatively primitive method for incrementing the subscript variable.

Now you've had a very quick walk through a very large and rich compiler. I hope you have gained at least an intuitive feel for the power of this unique language. This particular implementation of SNOBOL appears to be excellent. Its strengths are many and its weaknesses are relatively minor. ■

References

1. *The SNOBOL4 Programming Language*, 2nd Ed., Griswold, R.E., Poage, J. F., and Polonsky, I. P., Prentice-Hall Inc., 1971.
2. *A SNOBOL4 Primer*, Griswold, R. E. and Griswold, M. T., Prentice-Hall, Inc., 1973.

By Dan Daetwyler

```
Data types:  Integer, Real, String

     Integer:  +/- 32767
     Real:     +/- 9,007,199,254,740,991 (53 bit
                         mantissa - Intel)
     String:   32767 characters
     Arrays:   32767 elements
     Real to string:  16 digits
```

Table 1.

```
          &TRIM     = 1
          LOWER     = 'abcdefghijklmnopqrstuvwxyz'
          UPPER     = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
          WPAT      = BREAK(LOWER) SPAN(LOWER) . WORD
          COUNT     = TABLE(100,100)
          INPUT('IFILE',1,'SNO.REV','R,80')

READ      TEXT      = IFILE                       :F(PRINT)
          TEXT      = REPLACE(TEXT,UPPER,LOWER)
NEXTW     TEXT      WPAT =                        :F(READ)
          COUNT<WORD> = COUNT<WORD> + 1           :(NEXTW)

PRINT     RESULT    = CONVERT(COUNT,'ARRAY')      :F(NONE)
          OUTPUT    = 'WORD COUNT'
          I         = 1
PRTC      OUTPUT    = RESULT<I,1> ' -> ' RESULT<I,2> :F(END)
          I         = I + 1

NONE      OUTPUT    = 'THERE ARE NO WORDS'

END
```

Listing 1.

LOWER PROGRAMMING MAINTENANCE AND DEVELOPMENT COSTS

{SET:**SCIL**™}

The Source Code Interactive Librarian *for microcomputers.*

- **SCIL** keeps a historical record of all changes made to the library.
- **SCIL** maintains any source code regardless of language, including user documentation and text material.
- **SCIL** allows software engineers to work with source code as they do now, *using any ASCII text editor.*
- **SCIL** saves disk space by storing only the changes made to the program.
- **SCIL** provides a labeling capability for ease of maintaining multiple versions and multiple releases.
- **SCIL** offers unlimited description in the program library directory.
- *High visibility displays* with varied intensity for ease of viewing insertions and deletions.
- **SCIL** is available on CP/M, MP/MII, MS-DOS, PC-DOS and TurboDOS.

{SET} Get {SET} for Success
{SET:**SCIL**™} is a product of System Engineering Tools, Inc.
645 Arroyo Drive, San Diego, CA 92103

Registered Trademarks: CP/M, MP/M II - Digital Research Inc.; MS-DOS, Microsoft Corp.; PC-DOS, IBM Corp.; TurboDOS, Software 2000, Inc.

For more information call (619) 692-9464.

CIRCLE 64 ON READER SERVICE CARD

OPT-TECH SORT™

SORT/MERGE program for IBM-PC & XT

Now also sorts dBASE II files!

- Written in assembly language for **high performance**
  Example: 4,000 records of 128 bytes sorted to give key & pointer file in 30 seconds. **COMPARE!**
- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Handles variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- Output file can be full records, keys or pointers
- Can be run from keyboard or as a batch command
- Can be called as a subroutine to many languages
- Easy to use, includes on-line help feature
- Full documentation — sized like your PC manuals
- **$99** —VISA, M/C, Check, Money Order, COD, or PO
  Quantity discounts and OEM licensing available

To order or to receive additional information write or call:

OPT-TECH DATA PROCESSING
P.O. Box 2167 Humble, Texas 77347
(713) 454-7428

Requires DOS, 64K and One Disk Drive

CIRCLE 48 ON READER SERVICE CARD

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.
Alcor Systems	12	1
Alpha Computer Services	62	2
American Planning Corp.	10	3
Atron	2	4
BD Software	52	5
Borland International	1	6
Austin E. Bryant Consulting	62	7
Carousel Microtools	76	8
Catspaw, Inc.	54	9
Chromod Associates	73	10
The Code Works	74	11
CompuPro	Cover IV	12
Computer Innovations	60	13
Computer Resources of Waimea	74	14
Creative Solutions	27	15
C Systems	46	16
C User's Group	54	17
C Ware Corporation	72	18
Datalight	29	19
DWB Associates	9	20
Echelon, Inc.	25	21
Ecosoft	65	22
Foehn Consulting	54	23
Forth, Inc.	57	24
Forth, Inc.	59	25
Forth, Inc.	61	26
Forth Interest Group	68	27
Allen Gelder Software	54	28
Greenleaf Software	68	29
Hawkeye Grafix	12	30
HSC, Inc.	75	31
Introl Corporation	49	32
King Software	70	33
Korsmeyer Electronics Design Inc.	4	34
Laboratory Microsystems Inc.	42	35
Lattice, Inc.	70	36
Learning Tools, Inc.	35	37
Limbic Systems, Inc.	26	38
MBP Software & Systems Technology	53	39
MicroMotion	62	40
Miller Microcomputer Service	77	41
Morgan Computing	42	42
Mountain View Press	6	43
Mumps Users' Group	67	44
Next Generation Systems	41	45
Northwest Computer Algorythms	29	46
Opt-Tech Data Processing	71	47
Opt-Tech Data Processing	79	48
Parasec Research, Inc.	71	49
Plum Hall	69	50
Poor Person Software	76	51
The Programmers Shop	20	52
ProCode	65	53
Programmer's Connection	78	54
Quest Research, Inc.	Cover III	55
Rational Systems, Inc.	61	56
Robotics Age	30	57
RR Software, Inc.	Cover II	58
SLR Systems	26	59
Solution Systems	20	60
Solution Systems	20	61
Summit Software	4	62
Syntax Constructs Inc.	76	63
Systems Engineering Tools	79	64
Thunder Software	54	65
UniPress	74	66
Western Ware	54	67
Workman & Associates	48	68
WW Norton	48	69

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

ORDER THE PREMIER ISSUE OF COMPUTER LANGUAGE

The first issue of *COMPUTER LANGUAGE* was a great success and nearly sold out in just one month. We still have a few copies of this collectors edition available now. Just fill out this coupon and mail it back with $4.00 per issue.

Please send me _____ copies of COMPUTER LANGUAGE's premier issue at $4.00 per issue, $__________ Total

NAME ______________________________

COMPANY ______________________________

ADDRESS ______________________________

CITY, STATE, ZIP ______________________________

Send payment and coupon to:
COMPUTER LANGUAGE
Premier Issue
131 Townsend St.
San Francisco, CA 94107

ADVERTISE in the November issue of COMPUTER LANGUAGE

Reservation Deadline:
October 8th

Contact:
Carl Landau or Jan Dente
Computer Language
131 Townsend Street
San Francisco, CA 94107
(415) 957-9353

SUBSCRIBE TODAY!

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** at the Charter Subscription price today! Charter Subscription to **COMPUTER LANGUAGE** for only $19.95 — over 43% savings off the single copy price.

☐ Yes, start my Charter Subscription to COMPUTER LANGUAGE today. The cost is only $19.95 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $34.95.

☐ Payment enclosed ☐ Bill me

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Offer expires 12/84. Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Outside the U.S., add $12.00/year for surface mail or $30.00/year for airmail.

Canadian orders add $6.00 per year.

BI04

SUBSCRIBE TODAY!

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** at the Charter Subscription price today! Charter Subscription to **COMPUTER LANGUAGE** for only $19.95 — over 43% savings off the single copy price.

☐ Yes, start my Charter Subscription to COMPUTER LANGUAGE today. The cost is only $19.95 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $34.95.

☐ Payment enclosed ☐ Bill me

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Offer expires 12/84. Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Outside the U.S., add $12.00/year for surface mail or $30.00/year for airmail.

Canadian orders add $6.00 per year.

BI04

SUBSCRIBE TODAY!

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** at the Charter Subscription price today! Charter Subscription to **COMPUTER LANGUAGE** for only $19.95 — over 43% savings off the single copy price.

☐ Yes, start my Charter Subscription to COMPUTER LANGUAGE today. The cost is only $19.95 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $34.95.

☐ Payment enclosed ☐ Bill me

Name ______________________________

Company ______________________________

Address ______________________________

City, State, Zip ______________________________

Offer expires 12/84. Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Outside the U.S., add $12.00/year for surface mail or $30.00/year for airmail.

Canadian orders add $6.00 per year.

BI04

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

FREE!

Free information from the advertisers of **COMPUTER LANGUAGE.**

1. Please fill in your name and address on the card (one person to a card).
2. Answer questions 1-3.
3. Circle the numbers that correspond to the advertisements you are interested in.

READER SERVICE CARD

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Country ________ Telephone number ____________________

October issue. Not good if mailed after February 28, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Please complete these short questions:

1. I obtained this issue through:
   - ☐ Subscription
   - ☐ Computer Store
   - ☐ Retail outlet
   - ☐ Passed on by associate
   - ☐ Other ____________
2. Job Title ____________________
3. The 5 languages that I am most interested in reading about (list in order of importance).

____

____

Comments ____________________

____

____

Attn: Reader Service Dept.

FREE!

Free information from the advertisers of **COMPUTER LANGUAGE.**

1. Please fill in your name and address on the card (one person to a card).
2. Answer questions 1-3.
3. Circle the numbers that correspond to the advertisements you are interested in.

READER SERVICE CARD

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Country ________ Telephone number ____________________

October issue. Not good if mailed after February 28, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Please complete these short questions:

1. I obtained this issue through:
   - ☐ Subscription
   - ☐ Computer Store
   - ☐ Retail outlet
   - ☐ Passed on by associate
   - ☐ Other ____________
2. Job Title ____________________
3. The 5 languages that I am most interested in reading about (list in order of importance).

____

____

Comments ____________________

____

____

Attn: Reader Service Dept.

Reader suggestions

Editorial Response Card

We want to hear your comments and suggestions about this issue of **COMPUTER LANGUAGE.** Your reader feedback will enable us to provide you with the information you want. Thank you for your help!

Comments: ____________________

____

____

☐ Yes, I have an idea for a manuscript: ____________________

____

☐ Yes, I'm interested in reviewing technical manuscripts.

☐ Yes, I'm interested in reviewing software.

Name: ____________________

Company: ____________________

Address: ____________________

City, State, Zip: ____________________

Phone Number: ____________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 27346 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 27346 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

We thought about calling it MacSimplex . . . after all it makes your IBM®PC behave like a Macintosh™ and much more . . .

and with over two years in the making, the Simplex Database Management System has features like 32-megabyte virtual memory and the most powerful networked/relational database in the microcomputer industry. Simplex was designed around how you think and the Macintosh way, so that you can use your favorite mouse to handle those mundane tasks like menu selection and data manipulation. And, if you don't have a mouse, you can use our keyboard mouse simulator, MouSim™.

Pop-up and pull-down menus, dialog and alert boxes are not just added features, they are the heart of the Simplex way. In addition, Simplex gives you both a software and a hardware floating point capability, each with 19-digit accuracy. It permits login, password, privilege, and can be used on a local area network. Simplex has full communications and a remote or local printer spooler. Above all, Simplex is modular and grows with you! Simplex also has a full-featured, English-like language which is simple to use.

You can't buy Simplex™, but it is now available as an integral part of *it's my* ***Business***™ and will be used by *it's my* ***Word***™, *it's my* ***Graphics***™, . . .

Businessmen! *it's my* ***Business*** will revolutionize the way that you handle your business. It saves time, money, and standardizes your system for all who use it. *it's my* ***Business*** comes with applications like accounting, interoffice or intraoffice mail, editing, invoicing, inventory managment, mail list, calendar, scheduler, forms and more. You can modify each of these to create applications specifically designed for you... maybe we should have called it "it's your Business".

Professionals! *it's my* ***Business*** has over 200 pages of examples and demonstrations to show you how to solve your everyday professional problems. And if these examples aren't enough, we give you a complimentary one-year subscription to Questalk™, our hands-on Simplex applications magazine.

System integrators and consultants, beware! If you are not using *it's my* ***Business*** with Simplex to solve your problems, don't be surprised when more novice programmers solve that complex math, industrial engineering, or business problem faster. We think that you can cut your concept-to-development time by an order of magnitude!

it's my ***Business*** (includes *it's my* ***Editor***) - $695.00
it's my ***Business*** Demo Disk - $20.00
it's my ***Editor*** - $100.00.

Quest Research software is available through your local computer store or through mail order from Quest Software Corporation at (205) 539-8086. 303 Williams Avenue, Huntsville, AL 35801.

Value added resellers and dealers please contact Quest Research, Incorporated at (800) 558-8088. 303 Williams Avenue, Huntsville, AL, 35801.

Quest™

Quest Research Inc.

IBM is a registered trademark of International Business Machines. Macintosh is a trademark of Apple Corporation. *it's my* ***Business***, *it's my* ***Word***, *it's my* ***Graphics***, *it's my* ***Editor***, *it's my* ***Home***, *it's my* ***Voice***, *it's my* ***Ear***, *it's my* ***Statistics***, Simplex, MouSim, Questalk, and the Quest logo are trademarks of Quest Research, Incorporated.

CIRCLE 55 ON READER SERVICE CARD

HERE TODAY HERE TOMORROW

When buying a computer, you can't limit yourself to just satisfying today's needs. **The best value in a system comes from its productivity . . . both for today and tomorrow.** CompuPro's System 816™ computer has that value. With all the power and capacity to handle your needs now and down the road.

System 816's longevity stems from top quality components . . . high storage capacity . . . the flexibility to handle a large variety of applications . . . and the speed to get the job done fast. Upgrading is easy, and when it's time to expand from single to multi-user operation, it's as simple as plugging in boards and adding terminals. Your system grows as you grow.

CompuPro also provides a library of the most popular software programs with your system and because it's CP/M® based, you have more than 3,000 other programs to choose from.

Even our warranty is for today and tomorrow. It spans 365 days — and includes the additional security of Xerox Americare™ on-site service nationwide for designated systems.*

What's more, CompuPro is one company you can count on to be around tomorrow. For more than ten years we've been setting industry standards, increasing productivity and solving problems.

For a free copy of our business computer buyer's primer, and the location of the Full Service CompuPro System Center nearest you, call **(415) 786-0909 ext. 206.**

CompuPro's System 816. The computer that's just as essential tomorrow as it is today.

CompuPro®

A *GODBOUT* COMPANY

3506 Breakwater Court, Hayward, CA 94545

*Available from Full Service CompuPro System Centers and participating retailers only.
System 816 and The Essential Computer are trademarks of CompuPro.
CP/M is a registered trademark of Digital Research Inc. Americare is a trademark of Xerox Corporation.
System 816 front panel design shown is available from Full Service CompuPro System Centers only. ©1984 **CompuPro**

The Essential Computer™

CIRCLE 12 ON READER SERVICE CARD

SYSTEM 8/16
CompuPro.
CompuPro.
Hard Disk System